NUSDAS - METEOROLOGICAL DATABASE FOR OPERATIONAL WEATHER SERVICE

18.12

Eizi Toyoda*
University of Wisconsin-Madison, Madison, Wisconsin

1. INTRODUCTION

NuSDaS (Numerical Prediction Standard Dataset System) is a database system of Japan Meteorological Agency (JMA) that has been used in operational weather service since the main supercomputer of JMA was replaced with UNIX-based HITACHI SR8000 in 2001. Use of NuSDaS is mandatory in the JMA operational suites of numerical prediction, in order to standardize data format, data access subroutine, and terms of data structure.

NuSDaS was originally developed as a standalone database, which has no network functionality. HTTP-based remote data access extension is implemented afterwards. Using same API (application interface), an application program can access both local file and remote data with little limitations. This extension is also deployed in some experimental operations.

Currently a great deal of effort was being put into redesigning of file format and extension of API, aimed at the next replace of supercomputer of JMA planned in early 2006. Following argument will include the author's findings through experience, from which some improvements are planned.

2. DATA STRUCTURE

2.1 *Data Model*

Since NuSDaS is not general-purpose database, the data structure and API are highly specialized for meteorological data, especially those of numerical weather prediction. Database consists of data records, which are numeral (floating point or integer) data on a two-dimensional grid usually taken horizontally. A data record is identified by following identifiers:

1) data type name (16 character string),
2) reference time (initial time for forecast),
3) member name (of ensemble forecast; otherwise blank is used),
4) valid time,
5) level name, and
6) element name (parameter as in GRIB, also known as variable name).

Data type name identifies a dataset, and the others identify a data record in the dataset. In digital representation, reference time and valid time are integer in minutes from 0:00 UTC January 1, 1801, and the other identifiers are all character string with fixed length.

Data type name includes some description of dataset structure. Twelve of sixteen letters in the type name describe the nature of dataset (Appendix C.2.1), which loosely corresponds to Section I of GRIB Edition 1. However, it was quite unpopular that the snapshot quantity (such as temperature) and time-integrated quantity (such as precipitation) cannot be stored in one dataset, hence the time-integrated quantities are accommodated in the "guise" of snapshot quantities (see Appendix C.2.5 for detail).

Although some of the identifiers (reference time, valid time, and level, usually) are considered to be "dimension" in the viewpoint of physics, they are handled differently from horizontal (X-Y) dimensions, because NuSDaS must manage "allowed" combinations of "possible values" of identifiers. For example, element "RAIN" in three-dimensional forecast dataset is usually allowed only at surface level, and is not allowed at the initial time. This feature makes NuSDaS quite different from grid based data models such as NetCDF (Rew et al., 1997). However, there is urging demand of XYZ three dimensional I/O not only for convenience but

* Corresponding author address: Eiji TOYODA, Environmental Remote Sensing Center, University of Wisconsin-Madison, 1225 W Dayton St Atmospheric and Oceanic Science Bldg #1219, Madison, WI 53706; email: toyoda@wisc.edu. The author is dispatched from Japan Meteorological Agency.

also for performance, and there will be some modification to allow three (or possibly four) dimensional records.

2.2 *File Format*

A NuSDaS dataset is implemented as a directory tree with one definition file and data files. It is called NRD (NuSDaS rood directory).

Definition file (see Appendix A) is a plain text file that describes structure of NuSDaS dataset. Definition file must be prepared before creating dataset, and it will remain in the dataset to provide a quick-look of data structure for human administrators.

There are several variant formats for data file. The most typical one is similar to sequential file of Fortran. Unfortunately that has been incompatible since the record length field (4 octets) is larger by 8 than that in Fortran file, and the author plan to make it Fortran-compatible in the next release.

The data file (see Appendix B) consists of three kinds of records that are identified with four-letter label: records to define structure of file ("NUSD", "INDX", and "END"), metadata ("CNTL", "SUBC", and "INFO"), and DATA record. Records in the first group will be considerably redesigned since it caused the file size limitation of two gigabyte, which became inappropriate in today's supercomputing of telaflops processing and petabyte storage.

Data records are stored in separated data files for different data type, reference time and, by optional configuration, member or valid time. This feature is required since the operational suite has a rule that two or more batch jobs should not write onto one data file, in order to avoid loss of data when second or later job are aborted while writing file. Although this feature works well in most cases, the author and colleagues found difficulty in seasonal forecast, which has often nearly one thousand valid times.

3. APPLICATION INTERFACE (API)

The application interface (see Appendix C) for NuSDaS is provided for both C and Fortran. In comparison to general-purpose API's (such as NetCDF API), NuSDaS API is in higher level, or in other words, is designed for limited range of usage.

Data access routines (such as NUSDAS_READ) take arguments for record identifiers directly. Users do not have to convert character string (such as data type name or element name) into index. Although frequent conversion from string to index might cause inefficiency, the overhead has been negligible and improves readability of application program considerably.

Another user-friendly feature is that no initialization function call is needed before the data access routines. Unfortunately users must call output buffering flush routines (NUSDAS_ALLFILE_CLOSE etc.), if output buffering is enabled. Buffering improves output performance remarkably, and is mandatory in operational suites.

The API also has weakness. One example is that functions to browse dataset are poorly organized and incomplete. Hence tools for data browsing and manipulation, often have to bypass the API touching data files directly, and they tend to be ad-hoc or inefficient. The author advocates that further clarification of the data model is needed, and supposes the data model should be expressed as RDB-like collection of grid data.

4. IMPLEMENTATIONS

4.1 *Standalone Implementations*

The NuSDaS interface is written in C, including Fortran interface. Although the first version was designed only for HITACHI systems, recent implementations include configure script for automatic C-Fortran linkage and runtime automatic byte swapping.

There is another standalone implementation that is entirely written in Ruby, an object-oriented scripting language.

4.2 *Server-Client Implementations*

In order to facilitate data access from remote computers, a data transfer protocol using HTTP was developed (announced in Toyoda, 2002).

The protocol is designed so that Web browser can be used as data browser. The data server has functionalities of automatic data format conversion into not only numerical formats but also image files. The users are also able to browse metadata in several text formats.

Another new feature is traffic control. The server can relay a request for data to another server that has the data requested, and then the servers negotiate about which host convert the data format using content negotiation framework defined in HTTP.

4.3 *Conversion from/to Other Data Models*

Converter tools between GRIB (originally for edition 1, and later edition 2) and NuSDaS has been developed from the early history of NuSDaS. Although the data model of NuSDaS is largely influenced from that of GRIB, there is still a little discrepancy that makes it difficult to convert data automatically. Firstly, data identifiers and metadata have not always one-by-one correspondence (see note below Table C.2.1F for an example). Secondly, there is some expediential use of assigned name for identifiers (such as “snapshot guise” discussed in Section 2.1). Finally, users often willfully assign different name or units for elements.

Currently the author and colleagues have little experience in data exchange from/to other data models, such as NetCDF, OpenDAP, or GIS data standards. There are, however, many potential benefits, including time-saving in data handling and new visualization software. The author advocates that mutual comparison of data model and metadata standard is the key to further progress.

RREFERENCES

Toyoda, E., 2002: *Technical development for the management of a NWP routine system*. Proceedings of Tenth Workshop on the Use of High Performance Computing in Meteorology, 4-8 November 2002, ECMWF.
[PDF presentation is available at http://www.ecmwf.int/publications/library/ecpublications/proceedings/high_performance_computing_2002/toyoda_eizi.pdf]

Rew, R., D. Glenn, S. Emmerson, and H. Davies, 1997: *NetCDF User's Guide for C*. University Corporation for Atmospheric Research, Boulder, CO., pp. 149.

ACKNOWLEDGEMENT

Design and implementation of NuSDaS of the first version was mainly work of Mr. Norihisa Fujikawa. The network extension is made by collaboration with Mr. Masaki Hasegawa and Dr. Tabito Hara.

APPENDIX A: DEFINITION FILE FORMAT

NuSDaS definition file is a plain text file that describes structure of NuSDaS dataset. The definition file looks like free format. More precisely, the file is interpreted line by line. Each line is broken down to words by spaces (ASCII SPC). Note that TAB character is not allowed in the definition file. A line starting with keyword (listed below, case insensitive) starts statement. Following lines without keyword at the top of themselves are continued lines and interpreted as one statement with starting line. A line that begins with number sign ('**#**', also known as sharp or pound) is a comment line and will be safely ignored.

Statements can be omitted, unless noted 'mandatory'. There is a limitation in order of the statements. Since they are not (and cannot be easily) documented, the author recommends describing statements in order of following description.

A.1 *Definition Statements*

nusdas *version*

Specifies version of NuSDaS. If not omitted, it must be **10**. And if not omitted, it is highly recommended to place this statement at the top of definition file; in future versions of NuSDaS, there may be incompatible extension to the definition file, and this *version* will describe what version of NuSDaS you are using.

path ...

specifies the directory at which data files will be located. It is relative path from NRD. One of following syntax list is used for *words*.

path *relative_path template*

The relative path will be *template*. See Pathname Expansion for special symbols. By default this style is assumed, and "*/_model/_attribute/_space/_time/_name*" is used as *template*.

path nwp_path_s

Equivalent to statements "**path relative_path/_3d_name**" and "**filename _validtime**".

path nwp_path_vm

Equivalent to statements "**path relative_path/_3d_name**" and "**filename _member**".

path nwp_path_m

Equivalent to statements "**path relative_path/_3d_name/_member**" and "**filename _validtime**".

path nwp_path_bs

Equivalent to statements "**path relative_path/_3d_name/_basetime**" and "**filename _validtime**".

filename *filename*

Name of data file will be *filename*. Pathname expansion will be applied to *filename*. By default, **_basename** is assumed.

creator *creator*

Specifies information on creator of the data. It will written in NUSD record after prepending user name and host name.

type1 *_model _2d _3d*

This statement cannot be omitted. Word *_model* is four name characters (alphabet, number, and underline) representing model name or creation process. Word *_2d* is two name characters representing horizontal grid name. Word *_3d* is two name characters representing vertical grid name. See Appendix C for table of possible values.

type2 *_attribute _time*

This statement cannot be omitted. Word *_attribute* is two name characters representing data attribute. Word *_time* is two name characters representing time attribute. See Appendix C for table of possible values.

type3 *_name*

This statement cannot be omitted. Word *_name* is four name characters. You can use arbitrary name for this field; it does not affect behavior of library nor conventional meaning. Name "**STD1**" is used for the most typical operational dataset. **Note:** when *_name* is less than 4 letters, space character is appended in internal operation, such as contents of **CNTL** record. However, in Pathname Expansion, underline ("_")

character is **prefixed**.

member *n_dc inout*

Word *n_dc* is number of members (1 assumed by default). When *inout* is **in**, records for different members are stored in one file, and when *inout* is **out**, records for different members are stored in separated files.

memberlist *member member ...*

lists up members.

basetime *YYYYmmddHHMM*

This statement is **omitted in most cases.** It specifies base time. Format of *YYYYmmddHHMM* is same as **+%Y%m%d%H%M** in UNIX **date**(1) or **strftime**(3).

validtime *n_vt inout unit*

This statement cannot be omitted. This specifies number of valid times *n_vt* and *unit*, units of numbers in following **validtime1** and **validtime2** statements. Word *unit* should be one of **min, hour, day, pen, mon, week, jun**. When *inout* is **in**, records for different valid times are stored in one file, and when *inout* is **out**, records for different valid times are stored in separated files.

validtime1 arithmetic *initial step*

validtime1 *all_list vt1 vt2 vt3* ...

This statement cannot be omitted. At least and just one of above two formes should appear. This statement specifies list of the first part of valid time, called *valid1* in Application Interface. When the second word is **arithmetic**, the valid1 is an arithmetical series with specified *initial* and *step* value. When the second word is **all_list**, following words are interpreted as list of valid times. Usually the list is written in ascending order. All of the arguments *initial, step, vt1, ...* are in *unit*s declared in previous **validtime** statement.

validtime2 *ft1 ft2 ft3 ...*

validtime2 *-dt*

At least and just one of above two formes should appear. This statement specifies list of the second part of valid time, called *valid2* in Application Interface. When the former form is used, the list of valid2 will be (*vt1* + *ft1*), (*vt2* + *ft2*), (*vt3* + *ft3*), and so on. Usually the list is written in ascending order. When the latter form is used, the list of valid2 will be (*vt1* + *dt*), (*vt2* + *dt*), (*vt3* + *dt*), and so on. All of the arguments *dt, ft1, ft2, ...* are in *unit*s declared in previous **validtime** statement. If this statement is omitted, the special value **-1** is assumed as valid2.

plane *n_lv*

This statement cannot be omitted. Specifies the number of planes.

plane1 *name name name ...*

This statement cannot be omitted. Specifies the list of first plane. The list should have *n_lv* items. Usually the list is written in ascending order in height. It looks like descending order if pressure coordinate is used, (e.g. **SURF 1000 950 900 ...**).

plane2 *name name name ...*

Specifies the list of second plane. The list should have *n_lv* items. If this statement is omitted, the same list to that in **plane1** statement is assumed.

element *n_el*

This statement cannot be omitted. Specifies the number of elements.

elementmap *elemname elementmap*

This statement cannot be omitted, and will appear *n_el* times. It describes where is the element *elemname* allowed to write. See section Elementmap for detail.

size *nx ny*

This statement cannot be omitted, It indicates that the number of grid points is *nx* in X direction, and *ny* in Y direction. In most cases X is taken eastward and Y northward, although that is dependent to what coordinate system (*_2d* in **type1** statement) you use.

basepoint *ix iy lon lat*

This statement indicates that the location of grid numbered (*ix, iy*) is positioned (*lon, lat*). Both of *ix, iy* must be real number, *lon* must be real number with '**E**' or '**W**' appended, *lat* must be real number with '**N**' or '**S**' appended, Note that this statement is used with the geographical meaning shown above even if the 2D grid is taken vertically. In order to describe vertical grid point locations, SUBC record might be used.

distance *dx dy*

Indicates horizontal distance (in X and Y directions) between adjacent grid points. The units is degree when the grids are latitude-longitude grids, and is meter when map projection is applied. When the 2D grid is taken vertically, one of *dx, dy* shall be ignored. Note that the meridional grid distance *dy* is taken **southward**. It is positive in most JMA models: grid points with the smallest Y index are located at the northern end of 2D grid. On the contrary, if *dy* is negative, grid points with the smallest Y index are located at the southern end of 2D grid.

standard *lon lat lon2 lat2*

Specifies standard longitude/latitude. They are parameters of map projection, and only a part of them is used in some cases. It is dependent to horizontal grid style whether this statement is required or not. See following description of **others**.

others *lon3 lat3 lon4 lat4*

Specifies 3rd or 4th longitude/latitude. Meaning of parameters is dependent to projection. It is also dependent to horizontal grid style whether this statement is required or not.

in case of _2d=**LM**

The Lambert conformal projection has 3 parameters; use "**standard** *LoV Latin1 LoV Latin2*", where *LoV* is Y-axis longitude, and *Latin1* and *Latin2* is the first/second latitude where the secant cone cuts the earth. In most cases of JMA, it looks like "**standard 140.0E 30.0N 140.0E 60.0N**".

in case of _2d=**PS**

The polar stereographic projection has 2 parameters; use "**standard** *LoV LaD* **0E 0N**", where *LoV* is Y-axis longitude, and *LaD* is the latitude where grid point distance is defined. In most cases of JMA, it looks like "**standard 140.0E 30.0N 0E 0N**".

in case of _2d=**MR**

The Mercator projection has one parameter; use "**standard 0E** *LaD* **0E 0N**", where *LaD* is the latitude where grid point distance is defined.

in case of _2d=**OL**

The Lambert conformal projection has 3 parameters; use "**standard** *LoV Latin1 LoV Latin2*" and "**others** *LoP LaP RotAng***E 0N**", where *LoV* is Y-axis longitude, (*Latin1*, *Latin2*) is the first/second latitude where the secant cone cuts the earth, (*LoP*, *LaP*) is longitude/latitnude of the projection southern pole, and *RotAng* is the angle of rotation after projection. Unfortunately, the practice in JMA has been failed to write this parameter properly and you may have data with zero-filled corresponding fields (as in **2003-03-07**).

in case of other horizontal grids

Since there is no projection parameters, **standard** or **others** statements should not be written.

value *representation*

Describes how gridded data represents field. Word *representation* should be one of them:

value PVAL

values at grid point. This is the default.

value MEAN

average over volume/area around grid point

value REPR

representative value obtained with another method

packing *pack_mode*

Describes encoding scheme to be used in DATA record. See "Packing Type" in the Application Interface for table of possible values. By default, **2PAC** is assumed.

missing *miss_mode*

Describes how missing value is to be represented. Word *miss_mode* should be one of them:

missing NONE

There is no method for missing value in this case. This is the default.

missing UDFV

A certain value is missing value, and grids with the value should be regarded missing. see NUSDAS_PARAMETER_CHANGE for detail.

missing MASK

Grid points with valid data are indicated with bitmap for each DATA record. See NUSDAS_MAKE_MASK in the Application Interface for detail.

information *group filename*

If the definition file has this statement, INFO record will be written at the time of data file creation. It can be stated as many as needed. Size and contents of the INFO record will be that of file specified with a relative path *filename*. Word *group* should be a four-character name that identifies the INFO record.

subcntl *num group size group size* ...

If the definition file has this statement, SUBC record is allocated at the time of data file creation. Each SUBC record is secified with a pair of *group* (four-character name that identifies the SUBC record) and *size* (size of the SUBC record). Word *num* specifies the number of *group-size* pairs.

forcedlen *size*

This statement is required if you use ES interface. If the definition file has this statement, each records in data file will have *size* bytes. Padding of (*size* - (payload size)) bytes is used after record payload. Error occurs if a record exceeds the specified *size*. By default, records are aligned contiguously (without padding between record payload and 4-byte record trailer).

A.2 *Pathname Expansion*

Pathname of data file is determined by **path ...** and **filename** *filename* statements in the definition file, after substitution of following keywords to values of data identifier.

Table A.2: Keywords of Pathname Expansion

Keyword	Meaning
_model	model name, first 4 characters of type1
_2d	2D grid structure, 5th and 6th characters of type1
_3d	2D grid positioning, 7th and 8th characters of type1
_attibute	first two characters of type2
_time	time attribute, last two characters of type2
_name	type3
_space	equivalent to '_2d_3d'
_base	base time
_valid	valid time
_member	Member

Note that plane and element is not used in pathname expansion, since they cannot 'split' file. Similarly, using '**_valid**' or '**_member**' will cause malfunction if you declare '**valid** ... **in**' or '**member** ... **in**' respectively. On the other hand, if you declare '**valid** ... **out**' or '**member** ... **out**', you must use '**_valid**' or '**_member**' respectively in path ... or filename filename statements; otherwise data files for different valid times or members will collide (have same names and may cause malfunction).

A.3 *Elementmap*

Elementmap defines whether a certain element is allowed or not for certain combination of member, valid time, and plane. To understand elementmap, first think of a bitmap of size *M* * *V* * *P* (or Fortran logical array with DIMENSION(*P*, *V*, *M*)), where *M*, *V*, *P* are total number of members, valid times, and planes. For each bit, '**1**' declares that the element is allowed, and '**0**' does oppositely. Elementmap written in the definition file is the bitmap in a kind of run-length-encoding (RLE) compression.

The syntax of elementmap is written in BNF as follows:

elementmap := *member_loop* | *vtime_loop* | *vtime_line*
member_loop := 3 (*nmember* *member_block*)+
member_block := *vtime_loop* | *list_line*
vtime_loop := 2 (*nvtime* *bit_list*)+
vtime_line := *list_line* | *contiguous_line*
list_line := 1 *bit_list*
bit_list := ('**0**' | '**1**')+
contiguous_line := **0**
nmember := positive integer
nvtime := positive integer

They are interpreted as follows:

- *contiguous_line* tells that the element is allowed for **all** planes for certain valid time and member.
- *list_line* tells that the element is allowed only for planes indicated with symbol **1** in *bit_list*. The size of *bit_list* should be the number of planes.
- either *contiguous_line* or *list_line*, as *vtime_line*, can be used for *elementmap* even if the dataset has many valid times or members: it is assumed that the *contiguous_line* specification is repeated as many as needed.
- *vtime_loop* tells that the elementmap depends on the valid time. Sum of repeated *nvtime* numbers must be the number of valid times. If all *nvtime*'s are **1**, *bit_list* describes elementmap subarray for each valid time of corresponding order. When *nvtime* is more than 1, *bit_list* is treated as if it is repeated *nvtime* times.
- *member_loop* tells that the elementmap depends on the member. Sum of repeated *nmember* numbers must be the number of members. If all *nmember*'s are **1**, *member_block* describes elementmap subarray for each valid time of corresponding order. When *nmember* is more than 1, *member_block* is treated as if it is repeated *nmember* times.

The author admits the rule above is far from human understanding. Indeed, terms *vtime_loop* or *member_loop* are hardly used. **If you are not sure, declare elements with** *contiguous_line*. It will look like following:

```
element 4
elementmap PSEA 0
elementmap T 0
elementmap U 0
elementmap V 0
```

Allowing too much data records does not mean increase of data file size or data access speed/latency. Thus you can safely declare elements with 'no limitation' settings.

APPENDIX B: DATA FILE FORMAT

B.1 *Common Record Structure*

Common structure of records of NuSDaS data file is shown in the Table B.1.

Table B.1: Record Structure

Offset byte	Length Byte	Type	Description
0	4	Integer	*n*: record size
4	4	character	Kind of record
8	4	integer	*m*: payload size
12	4	integer	Creation date and time in **time_t** value
16	*m* – 8	—	PAYLOAD of record; see Table B.2–B.7 for detail
8 + *m*	*N* – *m* – 8	—	Padding; should be ignored
n – 4	4	integer	*n*: record size

Note that the `Type' is written in strange notation deliberately. They should NOT be directly interpreted as a type name of certain programming language, like C or Fortran.

character

Byte value should be interpreted as character code of ISO 646 IRV. Meaning of byte whose MSB is set is currently undefined.

integer

Certain number (usually 4) of bytes represents signed integer value. Negative value is represented with complement of 2. Note that **big endian** ordering of bytes is always used in NuSDaS data file.

unsigned integer

Certain number (usually 4) of bytes represents **unsigned** integer value.

floating

Bits in 4 or 8 bytes are used to compose IEEE 754 floating point value

.

Some field is array, and that is indicated in notation like C. For example, a field noted **character [2][*n_lv*][6]** is equivalent to memory image of **unsigned char [2][*n_lv*][6]** in C or **CHARACTER(LEN = 6), DIMENSION(*N_LV*, 2)** in Fortran. However, one-dimensional array notation '[size]' for scaler character field is omitted for simplicity.

Following tables describes various kinds of PAYLOAD part in Table B.1. Note that hereafter the symbol '∆' denotes a space (ASCII SPC).

B.2 *NUSD Record*

NUSD record conveys some metadata and information on file structure.

Table B.2: NUSD Record Format (only Payload shown)

Offset Byte	Length Byte	Type	Description
16	80	Character	creator host and user name.
96	4	Integer	NuSDaS version: currently 10
100	4	unsigned integer	total number of bytes in file
104	4	Integer	number of records in file
108	4	Integer	number of INFO records in file
112	4	Integer	number of SUBC records in file

B.3 *CNTL Record*

CNTL Record provides metadata mainly on georeference.

Table B.3: NuSDaS v1.0 CNTL Record Format (only Payload shown)

Offset	Length	Type	Description
Byte	Byte		
16	16	character	data type
32	12	character	base time in format like "date **+%Y%m%d%H%M**"
44	4	integer	base time in sequential minute from 1801-01-01T000Z
48	4	character	time unit for valid times
52	4	integer	*n_dc*: number of members
56	4	Integer	*n_vt*: number of valid times
60	4	Integer	*n_lv*: number of planes
64	4	Integer	*n_el*: number of elements
68	4	character	Map projection
72	2 * 4	Integer [2]	*nx* and *ny*: number of grid points in X and Y directions
80	2 * 4	Floating [2]	grid index of reference point
88	2 * 4	Floating [2]	latitude/longitude of reference point
96	2 * 4	Floating [2]	latitude/longitude distance between grid points
104	2 * 4	Floating [2]	1st standard latitude/longitude of map projection
112	2 * 4	Floating [2]	2nd standard latitude/longitude of map projection
120	2 * 4	Floating [2]	3rd standard latitude/longitude of map projection
128	2 * 4	Floating [2]	4th standard latitude/longitude of map projection
136	4	Character	PVAL: representation method of grid
140	2 * 4	---	reserved for future use of map projection
148	6 * 4	---	reserved for future use
172	*n_dc* * 4	character [*n_dc*][4]	list of member name
(1)	*n_vt* * 8	Integer [2][*n_vt*]	list of valid time pair
(2)	*n_lv* * 12	character [2][*n_lv*][6]	list of plane pair
(3)	*n_el* * 6	character [*n_el*][6]	List of element name

172 + 4 * *n_dc*

172 + 4 * *n_dc* + 8 * *n_vt*

172 + 4 * *n_dc* + 8 * *n_vt* + 12 * *n_lv*

B.4 *INDX Record*

Payload of INDX record is an array of 32bit unsigned integer whose size is *n_dc * n_vt * n_lv * n_el*. It provides byte offset of records in the datafile. Value zero indicates that the record is not written. Value **-1** indicates that the record is prohibited by elementmap statements.

B.5 *SUBC Record*

SUBC records convey various kind of metadata. They are subclassified with four-letter identifier.

B.5.1 *SUBC ETA/SIGM record*

This kind of SUBC record is employed to describe vertical grid structure. You can get pressure by *p*[*k*] = *b*[*k*] * (*p_surface* - *c*) + *a*[*k*], where *k* is the index of vertical plane and *p_surface* the surface pressure.

Table B.5.1: SUBC ETA/SIGM Record Format (only Payload shown)

Offset byte	Length Byte	Type	Description
16	4	Character	"**ETA∆**" or "**SIGM**"
20	4	Integer	number of planes
24	(*n_lv* + 1) * 4	float [*n_lv* + 1]	parameter a
...	(*n_lv* + 1) * 4	float [*n_lv* + 1]	parameter b
...	4	Float	parameter c

B.5.2 *SUBC Z* record*

Table B.5.2: SUBC Z* Record Format (only Payload shown)

Offset Byte	Length Byte	Type	Description
16	4	character	"**Z*∆∆**"
20	2 * 4	integer	nx and ny: number of grid points in X and Y directions
28	4	integer	number of planes
32	(*n_lv* + 1) * 4	float [*n_lv* + 1]	z-star location for each plane
...	4	Float	height of model top
...	(*nx* * *ny*) * 4	float [*nx* * *ny*]	Surface height

B.5.3 *SUBC TDIF record*

This kind of SUBC record is employed when time integration/average product (the *_attribute* should be **AV** or **MV**) is stored in a dataset for snapshot value (the *_attribute* is **SV**). The size of SUBC TDIF record depends on parameters *n_dc* (members) and *n_vt* described in CNTL record.

DATA records that refers this SUBC TDIF record has its element name beginning with underline ('_') in addition to that of the element name table.

Table B.5.2: SUBC TDIF Record Format (only Payload shown)

Offset byte	Length Byte	Type	Description
16	4	Character	"**TDIF**"
32	4 * *n_dc* * *n_lv*	integer [*n_dc*][*n_lv*]	dt (in seconds)
...	4* *n_dc* * *n_lv*	float [*n_dc*][*n_lv*]	span: integration time span (in seconds)

- the beginning of integration time is given by *valid1* + *dt* - *span*.
- the end of integration time is given by *valid1* + *dt*.

B.5.4 *SUBC RADR record*

This kind of SUBC record is used for datasets of radar observation. The size of SUBC RADR record depends on parameters *n_dc* (members), *n_vt*, *n_lv*, and *n_el* described in CNTL record.

Table B.5.4: SUBC RADR Record Format (only Payload shown)

Offset Byte	Length Byte	Type	Description
16	4	Character	"**RADR**"
32	4* *n_dc* * *n_vt* * *n_lv* * *n_el*	integer [*n_dc*][*n_vt*][*n_lv*][*n_el*]	Flags

Value of *flags* has these means:

0
ND.

1
Echo exists.

2
No echo exists.

3
No operation.

B.5.5 *SUBC ISPC record*

This kind of SUBC record is used for datasets of synthesized multiple radar observations.

The size of SUBC ISPC record depends on parameters *n_vt*, *n_lv*, and *n_el* described in CNTL record.

Table B.5.5: SUBC ISPC Record Format (only Payload shown)			
Offset	Length	Type	Description
byte	Byte		
16	4	Character	"**ISPC**"
32	512 * *n_vt* * *n_lv* * *n_el*	Integer [*n_vt*][*n_lv*][*n_el*][128]	Flags

B.6 *DATA Record*

DATA record is reclassified with its method of representing missing value.

B.6.1 *DATA NONE record*

DATA NONE record has no missing value.

Table B.6.1: DATA NONE Record Format (only Payload shown)			
Offset	Length	Type	Description
byte	Byte		
16	4	character	member name
20	8	integer [2]	valid times
28	12	character [2][6]	plane names
40	6	character	element name
46	2	---	Reserved
48	2 * 4	integer[2]	*nx* and *ny*: number of grid points in X and Y directions
56	4	Character	packing scheme such as "**2PAC**"
60	4	Character	"**NONE**"
64	...	...	PACKED DATA: see following description

B.6.2 *DATA UDVF record*

DATA UDVF record has one missing value. All grids with this missing value should be considered to be missing.

Table B.6.2: DATA UDFV Record Format (only Payload shown)			
Offset	Length	Type	Description
Byte	Byte		
16	4	character	member name
20	8	integer [2]	valid times
28	12	character [2][6]	Plane names
40	6	character	element name
46	2	---	Reserved
48	2 * 4	integer[2]	*nx* and *ny*: number of grid points in X and Y directions
56	4	character	packing scheme such as "2PAC"
60	4	character	"**UDFV**"
64	(various)	integer/floating	Missing value
...	...	...	PACKED DATA: see following description

B.6.3 *DATA MASK record*

DATA MASK record represents missing grids using a bitmap whose number of bits is equal to the number of grids.

Table B.6.3: DATA MASK Record Format (only Payload shown)			
Offset	Length	Type	Description
byte	Byte		
16	4	character	member name
20	8	integer [2]	valid times
28	12	character [2][6]	plane names
40	6	character	element name
46	2	---	Reserved
48	2 * 4	integer[2]	nx and ny: number

			of grid points in X and Y directions
56	4	character	packing scheme such as "**2PAC**"
60	4	character	"**MASK**"
64	4	integer	n_ms: number of bytes used for mask bitmap
68	N_ms	bitmap	mask bitmap
...	...	...	PACKED DATA: see following description

B.6.4 *Packed Data Format*

When the packing scheme is **1PAC**, **2PAC**, or **2UPC**, two 4-byte floating-point field *base* and *amp* is followed by an array of packed type. See Application Interface about the packed type. Unpacking is adding *base* after multiplying *amp*.

When the packing scheme is **4PAC**, it is similar to **2PAC** but *base* and *amp* is 8-byte floating-point value.

When the packing scheme is **RLEN**, three 4-byte integer field *nbit*, *maxv*, *num* is followed by octet stream containing compressed bit stream.

When the packing scheme is **GRIB**, the GRIB octet stream itself will be the packed data; although this feature is not implemented yet.

Otherwise, the packed data is array of packed type. Note that if the packing scheme is '**N1I2**' the packed value is 10 times of unpacked value.

B.7 *END Record*

END record is located at the end of data file. The contents are shown in Table 6.

Table B.7: NuSDaS v1.0 END Record Format (only Payload shown)			
Offset Byte	Length Byte	Type	Description
16	4	Unsigned integer	total number of bytes in file
20	4	Integer	Number of records in file

APPENDIX C: APPLICATION INTERFACE

C.1 *Function Interface*

- Application program is supposed to include “**nusdas.h**” in C, and “**nusdas_fort.h**” in Fortran.
- Symbol **N_SI4** in following text is typedef-ed name in "**nusdas.h**". It means 32bit signed integer type, and is equivalent to **int** in most environments.
- In Fortran, subroutine arguments given to place declared as **CHARACTER(*n*)** must have at least *n* characters length.
- In C, function arguments given to place declared as const char * must have at least the same size to corresponding Fortran interface. Character array have not to be terminated with NUL ('**¥0**') character.
- In NuSDaS interface functions/subroutines, the first 3-10 arguments and the last argument in Fortran have common meaning. They looks like as following.

```
/* C */
int nusdas_xxx(type1, type2, type3, btime,
memb, vtime1, plane1, elem, ...)
int nusdas_xxx2(type1, type2, type3,
btime, memb, vtime1, vtime2, plane1,
plane2, elem, ...)
        const char type1[8];
        const char type2[4];
        const char type3[4];
        N_SI4 *btime;
        const char memb[4];
        N_SI4 *vtime1;
        N_SI4 *vtime2;
        const char plane1[6];
        const char plane2[6];
        const char elem[6];
! Fortran
SUBROUTINE NUSDAS_XXX(type1,
type2, type3, btime, memb, vtime1, plane1,
elem, ..., iostat)
SUBROUTINE NUSDAS_XXX2(type1,
type2, type3, btime, memb, vtime1, vtime2,
plane1, plane2, elem, ..., iostat)
        CHARACTER(8):: type1
        CHARACTER(8):: type2
        CHARACTER(4):: type3
        INTEGER:: btime
```

```
CHARACTER(4):: memb
INTEGER:: vtime1
INTEGER:: vtime2
CHARACTER(6):: plane1
CHARACTER(6):: plane2
CHARACTER(6):: elem
INTEGER, INTENT(OUT):: iostat
```

Meaning of arguments are as follows.

Table C.1 Common arguments

symbol	Description
type1	Data type, 1st component
type2	Data type, 2nd component
type3	Data type, 3rd component
btime	Base time
memb	Member name
vtime1	Valid time, start
vtime2	Valid time, end
plane1	Plane name, upper
plane2	Plane name, lower
elem	Element name

Some subroutines/functions have two forms: name of one ends with '**2**', and the other without '**2**'. In such a case, the with-2-form is the essential interface. Calling the latter, the no-2-form, is equivalent to that of the with-2-form with *vtime2* and *plane2* set to **1** and *plane1* respectively.

C.1.1 *NUSDAS_READ*

Reads a two-dimensional array from NuSDaS dataset.

SYNOPSIS

```
/* C */
int nusdas_read(type1, type2, type3,
btime, memb, vtime1, plane1, elem, udata,
utype, usize)
int nusdas_read2(type1, type2, type3,
btime, memb, vtime1, vtime2, plane1,
plane2, elem, udata, utype, usize)
        void *udata;
        N_SI4 utype;
        N_SI4 usize;
! Fortran
SUBROUTINE NUSDAS_READ(type1,
type2, type3, btime, memb, vtime1, plane1,
elem, udata, utype, usize, iostat)
SUBROUTINE NUSDAS_READ2(type1,
type2, type3, btime, memb, vtime1, vtime2,
plane1, plane2, elem, udata, utype, usize,
iostat)
        ! udata may be any type
        INTEGER, INTENT(IN):: utype
        INTEGER, INTENT(IN):: usize
```

ARGUMENTS

The *vtime1* value **-1** can be used as wildcard: first record found in the dataset was returned.

udata

user data array, to which the requested data will be copied.

utype

see table of User Data Array Type.

usize

number of array elements allocated user data. It must equal to or be larger than that of dataset.

RETURN VALUE

> 0	Succeeded. The value is number of array elements obtained.
0	Warning: requested data record not written yet in existing data file.
-1	Warning: requested data file not created yet.
-2	Warning: "requested data is not registered yet".

-3	Error: inconsistency between CNTL and DATA records.
-4	Error: insufficient size of user array.
-5	Error: conversion to specified data type is not supported.
-6	Error: malformed run-length encoded data.
-7	Error: malformed CNTL or INDX record.
-10 to -99	(see error code table)

DESCRIPTION

NUSDAS_READ and **NUSDAS_READ2** reads a data record to user data array *udata*. If you are going to read data in a layer or a time span, use **NUSDAS_READ2**.

C.1.2 *NUSDAS_WRITE*

Writes a two-dimensional array into NuSDaS dataset.

SYNOPSIS

```
/* C */
int nusdas_write(type1, type2, type3,
btime, memb, vtime1, plane1, elem, udata,
utype, usize)
int nusdas_write2(type1, type2, type3,
btime, memb, vtime1, vtime2, plane1,
plane2, elem, udata, utype, usize)
        void *udata;
        N_SI4 utype;
        N_SI4 usize;
! Fortran
SUBROUTINE NUSDAS_WRITE(type1,
type2, type3, btime, memb, vtime1, plane1,
elem, udata, utype, usize, iostat)
SUBROUTINE NUSDAS_WRITE2(type1,
type2, type3, btime, memb, vtime1, vtime2,
plane1, plane2, elem, udata, utype, usize,
iostat)  ! udata may be any type
        INTEGER, INTENT(IN):: utype
        INTEGER, INTENT(IN):: usize
```

ARGUMENTS

udata

user data array, from which the requested data will be copied.

utype

see table of User Data Array Type.

usize

number of array elements allocated user data. It must equal to than that of dataset.

RETURN VALUE

> 0	Succeeded. The value is number of array elements written.
-2	Warning: requested data is not registered yet.
-3	Error: size of user data array is too small.
-4	Error: conversion to specified data type is not supported.
-5	Error: data record size exceeds fixed record size.
-6	Error: invalid missing value style.
-7	Error: mask bit pattern undefined.
-8	Error: overflow in packing.

-9	Error: invalid compression scheme.
-10 to -99	(see error code table)

DESCRIPTION

NUSDAS_WRITE and **NUSDAS_WRITE2** writes a data record from user data array *udata*. If you are going to write data in a layer or in a time span, use **NUSDAS_WRITE2**.

C.1.3 *NUSDAS_ALLFILE_CLOSE*

SYNOPSYS

```
/* C */
int nusdas_allfile_close(closemode)
        int closemode;
! Fortran
SUBROUTINE
NUSDAS_ALLFILE_CLOSE(closemode,
iostat)
        INTEGER:: clodemode
```

ARGUMENTS

closemode

one of following symbols:

N_FOPEN_READ

all files opened as read-only are closed.

N_FOPEN_WRITE

all files opened as writable are closed.

N_FOPEN_ALL

all files are closed.

RETURN VALUE

> 0	Succeeded. The value is number of files successfully closed.
< 0	Error: the absolute value is number of files that cause error.

C.1.4 *NUSDAS_ESF_FLUSH*

flushes write buffer to ES files and close it.

SYNOPSIS

```
/* C */
int nusdas_esf_flush(type1, type2, type3,
btime, memb, vtime1);
int nusdas_esf_flush2(type1, type2,
type3, btime, memb, vtime1, vtime2);
! Fortran
SUBROUTINE
NUSDAS_ESF_FLUSH(type1, type2,
type3, btime, memb, vtime1, iostat)
SUBROUTINE
NUSDAS_ESF_FLUSH2(type1, type2,
type3, btime, memb, vtime1, vtime2, iostat)
```

C.1.5 *NUSDAS_GRID*

SYNOPSIS

```
/* C */
int nusdas_grid(type1, type2, type3,
btime, memb, vtime1, proj, gridsize,
geometry, tattr, io)
int nusdas_grid2(type1, type2, type3,
btime, memb, vtime1, vtime2, proj, gridsize,
geometry, tattr, io)
        char proj[4];
        N_SI4 gridsize[2];
        float geometry[7][2];
        char tattr[4];
        char io[4];
SUBROUTINE NUSDAS_GRID(type1,
type2, type3, btime, memb, vtime1, proj,
gridsize, geometry, tattr, io, iostat)
SUBROUTINE NUSDAS_GRID2(type1,
type2, type3, btime, memb, vtime1, vtime2,
proj, gridsize, geometry, tattr, io, iostat)
        CHARACTER(4):: proj
        INTEGER:: gridsize(2)
        REAL:: geometry(2, 7)
        CHARACTER(4):: tattr
        CHARACTER(4):: io
```

ARGUMENTS

proj

four-character symbol of projection type. The name is different from that in type1 string, and listed in Table C.1.5.

gridsize

grid size.

geometry

byte offset 80--135 in CNTL record.

tattr

four-character symbol which denotes how grid point value represents physical field with regard to spatial dimensions. Note that it is different from symbols in *type2* string. See value statement in the definition file description for possible values.

io

one of following symbols

N_IO_PUT

writes SUBC record.

N_IO_GET

reads SUBC record.

RETURN VALUE

0	Succeeded.
-1	Error: requested file not exist.
-2	Error: requested file malformed.
-5	Error: invalid argument.
-10 to -99	(see error code table)

C.1.6 *NUSDAS_INQ_CNTL*

SYNOPSIS

```
/* C */
int nusdas_inq_cntl(type1, type2, type3, btime, memb, vtime1, param, value, vsize)
int nusdas_inq_cntl2(type1, type2, type3, btime, memb, vtime1, vtime2, param, value, vsize)
        int param;
        void *value;
        N_SI4 *vsize;
! Fortran
SUBROUTINE NUSDAS_INQ_CNTL(type1, type2, type3, btime, memb, vtime1, param, value, vsize, iostat)
SUBROUTINE NUSDAS_INQ_CNTL2(type1, type2, type3, btime, memb, vtime1, vtime2, param, value, vsize, iostat)
        INTEGER:: param
        ! value may be different type
        INTEGER:: visze
```

DESCRIPTION

Meaning of arguments is same to that of NUSDAS_INQ_DEF. However, only following parameter symbols are acceptable: **N_MEMBER_NUM**, **N_MEMBER_LIST**, **N_VALIDTIME_NUM**, **N_VALIDTIME_LIST**, **N_PLANE_NUM**, **N_PLANE_LIST**, **N_ELEMENT_NUM**, and **N_ELEMENT_LIST**. See also NUSDAS_GRID to get grid information from CNTL record.

RETURN VALUE

>= 0	Succeeded. Value is number of array elements retrieved.
-1	Error: size of value array insufficient.
-2	Error: value array not allocated
-3	Error: invalid argument.
-10 to -99	(see error code table)

C.1.7 *NUSDAS_INQ_DATA*

Note that this function is not provided in the first version.

SYNOPSIS

```
/* C */
int nusdas_inq_data(type1, type2, type3, btime, memb, vtime1, plane1, elem, param, value, vsize)
int nusdas_inq_data2(type1, type2, type3, btime, memb, vtime1, vtime2, plane1, plane2, elem, param, value, vsize)
! Fortran
SUBROUTINE NUSDAS_INQ_DATA(type1, type2, type3, btime, memb, vtime1, plane1, elem, param, value, vsize, iostat)
SUBROUTINE NUSDAS_INQ_DATA2(type1, type2, type3, btime, memb, vtime1, vtime2, plane1, plane2, elem, param, value, vsize, iostat)
```

DESCRIPTION

This subroutine inquires metadata about the DATA record. Use NUSDAS_INQ_CNTL or NUSDAS_INQ_DEF to inquire about CNTL record, and use NUSDAS_INQ_DEF about definition file.

ARGUMENTS

param

may be following values

N_GRID_SIZE

Grid size of the DATA record will be stored at *value*.

N_PC_PACKING

Four letters indicating Packing Type will be stored at *value*. Note that the symbol **N_PACKING** is absent in header file of the operational version.

N_MISSING_MODE

Four letters indicating missing value handling will be stored at *value*. Possible values are **NONE**, **UDFV**, or **MASK**. See missing statement of the definition file description. Note that the symbol **N_PACKING** is absent in header file of the operational version.

N_MISSING_VALUE

Missing value of the DATA record will be stored at *value*. This query is available only when previous query of **N_MISSING_MODE** returns **UDFV**.

value

array to which parameter will be stored.

vsize

number of elements (not byte size) of array value.

RETURN VALUE

>= 0	Succeeded. Value is number of array elements retrieved.
-1	Error: size of value array insufficient.
-2	Error: value array not allocated
-3	Error: invalid argument.
-10 to -99	See error code table

C.1.8 *NUSDAS_INQ_DEF*

SYNOPSIS

```
/* C */
int nusdas_inq_def(type1, type2, type3, param, value, vsize)
        void *value;
        N_SI4 vsize;
! Fortran
SUBROUTINE NUSDAS_INQ_DEF(type1, type2, type3, param, value, vsize, iostat)
        ! value may be any type
        INTEGER:: vsize
```

ARGUMENTS

param

one of following symbols:

N_MEMBER_NUM

number of members. *value* is assumed to be INTEGER.

N_MEMBER_LIST

list of members. that is assumed to be array of CHARACTER(4).

N_VALIDTIME_NUM

number of valid times. *value* is assumed to be INTEGER.

N_VALIDTIME_LIST

list of forecast time (offset of valid time from base time). The *value* is assumed to be array of INTEGER. Note that the value is **NOT ALWAYS IN MINUTES**.

N_VALIDTIME_UNIT

four-letter symbol of unit of time, which is used in interpreting **N_VALIDTIME_LIST**. *value* is assumed to be CHARACTER(4).

N_PLANE_NUM

N_PLANE_LIST

N_ELEMENT_NUM

N_ELEMENT_LIST

N_ELEMENT_MAP

N_PROJECTION

N_GRID_SIZE

N_GRID_DISTANCE

N_GRID_BASEPOINT

N_STAND_LATLON

N_SPARE_LATLON

value

array to which parameter will be stored.

vsize

number of elements (not byte size) of array *value*.

RETURN VALUE

>= 0	Succeeded. Value is number of array elements retrieved.
-1	Error: size of value array insufficient.
-2	Error: value array not allocated
-3	Error: invalid argument.
-10 to -99	See error code table

C.1.9 *NUSDAS_INQ_NRDBTIME*

get list of basetime

SYNOPSIS

```
/* C */
int nusdas_inq_nrdbtime(type1, type2,
type3, bdata, bsize, verbose)
        N_SI4 *bdata;
        N_SI4 *bsize;
        ! verbose should be constant macro
defined in the header file
! Fortran
SUBROUTINE
NUSDAS_INQ_NRDBTIME(type1, type2,
type3, bdata, bsize, verbose, iostat)
        INTEGER:: bsize
        INTEGER:: bdata(bsize)
```

ARGUMENTS

bdata

array to which the list of base time will be stored.

bsize

element number (not byte size) of array bdata

verbose

one of following symbols:

N_ON

message is output while search is proceeding

N_OFF

message is suppressed

RETURN VALUE

>= 0	Succeeded. Value is number of array elements retrieved.
-1	Error: size of value array insufficient.
-2	Error: value array not allocated
-3	Error: invalid argument.
-4	Error: cannot open file or directory.
-10 to -99	See error code table.

LIMITATION

- Current implementation cannot handle dataset which has more base times than 999.
- The first version of this function does not work in some path styles.

C.1.10 *NUSDAS_INQ_NRDVTIME*

Get list of valid time in specified dataset.

```
/* C */
int nusdas_inq_nrdvtime(type1, type2, type3, bdata, bsize, verbose)
        N_SI4 *bdata;
        N_SI4 *bsize;
        ! verbose should be constant macro defined in the header file
! Fortran
SUBROUTINE NUSDAS_INQ_NRDVTIME(type1, type2, type3, bdata, bsize, verbose, iostat)
        INTEGER:: bsize
        INTEGER:: bdata(bsize)
```

ARGUMENTS

bdata

array to which the list of valid time will be stored.

bsize

element number (not byte size) of array *bdata*

verbose

one of following symbols:

N_ON

message is output while search is proceeding

N_OFF

message is suppressed

RETURN VALUE

See NUSDAS_INQ_NRDBTIME.

LIMITATION

The first version does not support this function.

C.1.11 *NUSDAS_IOCNTL*

SYNOPSIS

```
/* C */
int nusdas_iocntl(param, value)
! Fortran
SUBROUTINE NUSDAS_IOCNTL(param, value, iostat)
```

ARGUMENTS

param

Should be one of following symbols defined in header file.

N_IO_MARK_END

If *value* is **N_ON**, the END record is written to data file for each call of NUSDAS_WRITE functions. That is the default; it is safer, but slower. If *value* is **N_OFF**, the END record is written only when the data file is closed. Though it makes NUSDAS_WRITE faster, files may be corrupt and unreadable if writing program exits without closing the file. Note that **N_ON** is assumed when parameter **N_IO_W_FCLOSE** is set to **N_ON**.

N_IO_W_FCLOSE

If *value* is **N_ON**, data file opened for write is closed for each call of NUSDAS_WRITE functions. That is the default; it is safer, but slower. If *value* is **N_OFF**, such data files will be closed only when either NUSDAS_ONEFILE_CLOSE or NUSDAS_ALLFILE_CLOSE is called explicitly. Though it makes NUSDAS_WRITE faster, files may be corrupt and unreadable if writing program exits without closing the file. And you will also have to care about shortage of file handle resource: if too many files are opened simultaneously, further open may fail.

N_IO_R_FCLOSE

If *value* is **N_ON**, data file opened for read is closed for each call of NUSDAS_WRITE functions. That is the default; it is safer, but slower. If value is **N_OFF**, such data files will be closed only when either NUSDAS_ONEFILE_CLOSE or NUSDAS_ALLFILE_CLOSE is called explicitly. You will have to care about shortage of file handle resource.

N_IO_WARNING_OUT

If *value* is **N_ON**, subroutines in the NuSDaS interface output messages on waring/error to the standard error handle. That is the default. If value is **N_OFF**, such messages are suppressed.

value

Should be **N_ON** or **N_OFF** defined in header file.

RETURN VALUE

NUSDAS_IOCNTL returns **0** when succeeded, or **-1** when error (invalid argument).

C.1.12 *NUSDAS_MAKE_MASK*

SYNOPSIS

```
/* C */
int nusdas_make_mask(udata, utype,
usize, bitmap, bitmapsize)
        void *udata;
        N_SI4 *usize;
        char *bitmap;
        N_SI4 *bitmapsize;
! Fortran
SUBROUTINE
NUSDAS_MAKE_MASK(udata, utype,
usize, bitmap, bitmapsize, iostat)
        ! udata may be any type
        INTEGER:: usize
        CHARACTER:: INTENT(OUT)::
bitmap(*)
        INTEGER:: bitmapsize
```

ARGUMENTS

udata

user data array from which the bitmap is created. Array elements where data is missing should be set to the missing value for the User Data Array Type. For example, elements equivalent to **N_MV_R4** are regarded as missing if *udata* is REAL/float array.

utype

see table of User Data Array Type.

usize

number of array elements allocated user data.

bitmap

character array to which the bitmap will be stored.

bitmapsize

size of bitmap in bytes.

RETURN VALUE

NUSDAS_MAKE_MASK returns **0** when succeeded, or **-1** when error (bitmap size too small).

C.1.13 *NUSDAS_ONEFILE_CLOSE*

SYNOPSIS

/* C */
int **nusdas_onefile_close**(*type1, type2, type3, btime, memb, vtime1*)
int **nusdas_onefile_close2**(*type1, type2, type3, btime, memb, vtime1, vtime2*)
! Fortran
SUBROUTINE **NUSDAS_ONEFILE_CLOSE**(*type1, type2, type3, btime, memb, vtime1, iostat*)
SUBROUTINE **NUSDAS_ONEFILE_CLOSE2**(*type1, type2, type3, btime, memb, vtime1, vtime2, iostat*)

RETURN VALUE

1	Specified file is not opened.
0	Succeeded.
-2	Warning: requested data is not registered yet.
-3	Error: inconsistency between CNTL and DATA records.
-4	Error: array size too large.
-5	Error: array type mismatch.
-6	Error: too large value in RLE compression.
-7	Error: malformed CNTL or INDX record.
-10 to -99	(see error code table)

DESCRIPTION

NUSDAS_ONEFILE_CLOSE closes one data file specified with the arguments. It should be called before the end of the calling program which have called NUSDAS_WRITE series and have cleared **N_IO_W_FCLOSE** flag using NUSDAS_IOCNTL.

C.1.14 *NUSDAS_PARAMETER_CHANGE*

This function changes one of parameters of NuSDaS. Note that all parameters have global effect: once some parameter is changed, the effect remains until the end of calling process, even when different data type is accessed. Take care of yourself, especially on changing **N_PC_PACKING**, **N_PC_SIZEX**, and **N_PC_SIZEY**.

SYNOPSIS

/* C */
int **nusdas_parameter_change**(*param, value*)
! Fortran
SUBROUTINE **NUSDAS_PARAMETER_CHANGE**(*param , value, iostat*)

ARGUMENTS

param

specifies what parameter to change. It should be one of them:

N_PC_MISSING_UI1

missing value when user data array is char. The *value* should be (pointer to) the same type. The default value is **UCHAR_MAX**.

N_PC_MISSING_SI2

missing value when user data array is short int (INTEGER*2). The *value* should be (pointer to, in C) the same type. The default value is **SHRT_MIN**.

N_PC_MISSING_SI4

missing value when user data array is integer. The *value* should be (pointer to) the same type. The default value is **LONG_MIN**.

N_PC_MISSING_R4

missing value when user data array is float. The *value* should be (pointer to) the same type. The default value is **FLT_MAX**.

N_PC_MISSING_R8

missing value when user data array is double. The default value is **DBL_MAX**. The *value* should be (pointer to) the same type.

N_PC_MASK_BIT

bitmap for missing grid points. The *value* should be created as *bitmap* of

NUSDAS_MAKE_MASK. Default value for the parameter is meaningless, since writing without initializing bitmap fails (if bitmap required), or otherwise simply ignored.

N_PC_ID_SET

NRD number to/from which further access is done. This is useful in switching among many NRD's providing same data type. **N_OFF** is the default.

N_PC_PACKING

packing scheme. See the packing type table for acceptable values. **N_OFF** is the default.

N_PC_SIZEX

grid size in X direction. **N_OFF** is the default.

N_PC_SIZEY

grid size in Y direction. **N_OFF** is the default.

value

Value to be set as parameter. If **NULL** is given, it is equivalent to the default value. If you are using Fortran interface, be careful to tricky implementation of **NULL** in **nusdas_fort.h** due to consideration to compilers that does not support Fortran 95. It will override the built-in function **NULL** of Fortran 95 standard, hence subroutines using NuSDaS interface is not allowed to use **NULL** function.

C.1.15 *NUSDAS_SUBC_ETA*

Setting and reference of SUBC metadata record for eta vertical coordinate system.

SYNOPSYS

```
/* C */
int nusdas_subc_eta(type1, type2, type3,
btime, memb, vtime1, nlev, a, b, c, io)
int nusdas_subc_eta2(type1, type2,
type3, btime, memb, vtime1, vtime2, nlev,
a, b, c, io)
        N_SI4 *nlev;
        float *a;
        float *b;
        float *c;
! Fortran
SUBROUTINE
NUSDAS_SUBC_ETA(type1, type2, type3,
btime, memb, vtime1, nlev, a, b, c, io,
iostat)
SUBROUTINE
NUSDAS_SUBC_ETA2(type1, type2,
type3, btime, memb, vtime1, vtime2, nlev,
a, b, c, io, iostat)
        INTEGER nlev;
        REAL:: a(nlev);
        REAL:: b(nlev);
        REAL:: c(nlev);
```

DESCRIPTION

Features of this function, including arguments and return value, is the same to those of NUSDAS_SUBC_SIGM, except for the function name.

C.1.16 *NUSDAS_SUBC_INFO*

SYNOPSIS

```
/* C */
int nusdas_subc_info(type1, type2, type3, btime, memb, vtime1, infogroup, idata, isize, io)
int nusdas_subc_info2(type1, type2, type3, btime, memb, vtime1, vtime2, infogroup, idata, isize, io)
        char infogroup[4];
        char *iodata;
        N_SI4 *isize;
! Fortran
SUBROUTINE NUSDAS_SUBC_INFO(type1, type2, type3, btime, memb, vtime1, infogroup, idata, isize, io, iostat)
SUBROUTINE NUSDAS_SUBC_SRF2(type1, type2, type3, btime, memb, vtime1, vtime2, infogroup, idata, isize, io, iostat)
        CHARACTER(4):: infogroup
        CHARACTER:: iodata(isize)
        INTEGER:: isize
```

ARGUMENTS

infogroup

four-letter string that identifies the INFO record in the data file.

io

one of following symbols

N_IO_PUT

It makes the function write SUBC record.

N_IO_GET

It makes the function read SUBC record.

RETURN VALUE

>= 0	Succeeded. Value is size of INFO record retrieved.
-1	Error: requested file not exist.
-2	Error: requested file malformed.
-5	Error: invalid argument.
-10 to -99	(see error code table)

C.1.17 *NUSDAS_SUBC_PRESET1*

defines the default content of SUBC records for eta and sigma coordinates.

SYNOPSIS

```
/* C */
int nusdas_subc_preset1(type1, type2, type3, subcgroup, nlev, a, b, c)
        char subcgroup[4];
        N_SI4 *nlev;
        float *a, *b, *c;
! Fortran
SUBROUTINE NUSDAS_SUBC_PRESET1(type1, type2, type3, subcgroup, nlev, a, b, c, iostat)
        CHARACTER(4):: subcgroup
        INTEGER:: nlev
        REAL:: a, b, c
```

ARGUMENTS

subcgroup

must be "**SIGM**" or "**ETA∆**".

nlev

number of levels

a

b

c

vertical coordinate parameters.

RETURN CODE

0	Succeded
-1	invalid subcgroup value
-2	Too many SUBC records written.
-10 to -99	(see error code table)

C.1.18 *NUSDAS_SUBC_SIGM*

Setting and reference of SUBC metadata record for sigma vertical coordinate system.

SYNOPSYS

```
/* C */
int nusdas_subc_sigm(type1, type2, type3, btime, memb, vtime1, nlev, a, b, c, io)
int nusdas_subc_sigm2(type1, type2, type3, btime, memb, vtime1, vtime2, nlev, a, b, c, io)
        N_SI4 *nlev;
        float *a;
        float *b;
        float *c;
! Fortran
SUBROUTINE NUSDAS_SUBC_SIGM(type1, type2, type3, btime, memb, vtime1, nlev, a, b, c, io, iostat)
SUBROUTINE NUSDAS_SUBC_SIGM2(type1, type2, type3, btime, memb, vtime1, vtime2, nlev, a, b, c, io, iostat)
        INTEGER nlev;
        REAL:: a(nlev);
        REAL:: b(nlev);
        REAL:: c(nlev);
```

ARGUMENTS

nlev

number of levels

a

b

c

vertical coordinate parameters

io

one of following symbols

N_IO_PUT

writes SUBC record made from *a*, *b*, and *c*

N_IO_GET

reads SUBC record to *a*, *b*, and *c*

RETURN VALUE

0	Succeeded.
-1	Error: requested file not exist.
-2	Error: requested record not exist.
-3	Error: specified size is different to that of definition file.
-4	Error: specified number of level is smaller than that in file.
-5	Error: invalid argument.
-10 to -99	(see error code table)

C.1.19 *NUSDAS_SUBC_SRF*

```
/* C */
int nusdas_subc_srf(type1, type2, type3, btime, memb, vtime1, plane1, elem, subcgroup, idata, io)
int nusdas_subc_srf2(type1, type2, type3, btime, memb, vtime1, vtime2, plane1, plane2, elem, subcgroup, idata, io)
          char infogroup[4];
          N_SI4 *idata;
! Fortran
SUBROUTINE NUSDAS_SUBC_SRF(type1, type2, type3, btime, memb, vtime1, plane1, elem, subcgroup, idata, io, iostat)
SUBROUTINE NUSDAS_SUBC_SRF2(type1, type2, type3, btime, memb, vtime1, vtime2, plane1, plane2, elem, subcgroup, idata, io, iostat)
          CHARACTER(4):: infogroup
          INTEGER:: iodata(*)
```

ARGUMENTS

subcgroup

must be "**RADR**" or "**ISPC**"

io

one of following symbols

N_IO_PUT

writes SUBC record.

N_IO_GET

reads SUBC record.

RETURN VALUE

0	Succeeded.
-1	Error: requested file not exist.
-2	Error: requested record not exist.
-3	Error: specified size is different to that of definition file.
-4	Error: invalid subc group name.
-5	Error: invalid argument.
-10 to -99	(see error code table)

C.1.20 *NUSDAS_SUBC_TDIF*

SYNOPSIS

```
/* C */
int nusdas_subc_tdif(type1, type2, type3, btime, memb, vtime1, dtime, atime, io)
int nusdas_subc_tdif2(type1, type2, type3, btime, memb, vtime1, vtime2, dtime, atime, io)
          N_SI4 *atime, *dtime;
! Fortran
SUBROUTINE NUSDAS_SUBC_TDIF(type1, type2, type3, btime, memb, vtime1, dtime, atime, iostat)
SUBROUTINE NUSDAS_SUBC_TDIF2(type1, type2, type3, btime, memb, vtime1, vtime2, dtime, atime, iostat)
          INTEGER:: atime(*), dtime(*)
```

ARGUMENTS

dtime

difference of real valid time from nominal valid time in minutes.

atime

accumulation time in seconds.

io

one of following symbols

N_IO_PUT

writes SUBC record.

N_IO_GET

reads SUBC record

RETURN VALUE

0	Succeeded.
-1	Error: requested file not exist.
-2	Error: requested record not exist.
-3	Error: specified size is different to that of definition file.
-10 to -99	(see error code table)

C.2 *Tables*

C.2.1 *Data Identifier*

MODEL NAME

This horizontal grid name is used as 1st through 4th letter of function argument type1 or *_model* for type1 statement in the definition file.

Table C.2.1A: Model Name	
Value	Model description
_GSM	global spectral model
_RSM	regional spectral model
_MSM	Mesoscale model
_TYM	typhoon model
_DCD	decoded observation bulletins
_SRF	very short-range precipitation forecast
_WFM	week-range ensemble model
_SF1	month-range ensemble model
_SF4	four-month-range ensemble model
_XXX	(reserved for data from unspecified creation process)

HORIZONTAL GRID NAME

This horizontal grid name is used as 5th and 6th letter of function argument type1 or *_2d* for type1 statement in the definition file.

Table C.2.1B: Horizontal Grid Name	
Name	Description
LL	latitude-longitude regular grid
LM	Lambert conformal projection
PS	Polar Stereographic projection
GS	Gaussian grids
MR	Mercator projection
OL	Oblique-axis (rotated) lambert conformal projection
RD	Local Cartesian coordinate for radar
ST	Station data
YP	Meridional vertical cross section using pressure coordinate
XP	East-west vertical cross section using pressure coordinate
XX	(reserved for data on unknown grids)

VERTICAL GRID NAME

This horizontal grid name is used as 7th and 8th letter of function argument type1 or *_3d* for type1 statement in the definition file.

Table C.2.1C: Vertical Grid Name	
name	Description
PP	isobaric plane/layer
ET	eta vertical coordinate plane/layer
SG	sigma vertical coordinate plane/layer
HB	hybrid vertical coordinate plane/layer
LA	latitude (for vertical cross section data)
LO	longitude (for vertical cross section data)
ZZ	plane/layer specified by height
TH	isentropic plane/layer
ZS	Z* vertical coordinate plane/layer
XX	(reserved for data on unknown grids)

DATA ATTRIBUTE NAME

This horizontal grid name is used as the first two letter of function argument type2 or *_attribute* for type2 statement in the definition file.

Table C.2.1D: Data Attribute Name	
name	Description
FC	Forecast
EA	early analysis
AA	cycle analysis
RA	Reanalysis
CC	Constants
OB	Observations
GS	Guess
XX	(reserved for data of unknown nature

TIME ATTRIBUTE NAME

This horizontal grid name is used as the 3rd and 4th letter of function argument type2 or *_time* for type2 statement in the definition file. When this value is "**MV**" or "**AV**", special conventions are applied: please see notes on element name table.

Table C.2.1E: Time Attribute Name	
name	Description
SV	snapshot: the valid time has only one significant value
MV	time-average value for time span specified by *valid1* and *valid2*
AV	summation value for time span specified by *valid1* and *valid2*
DV	Standard deviation value for region specified by *valid1* and *valid2*
...	...
XX	(reserved for data of unknown nature)

PLANE NAME

This horizontal grid name is used as the function argument plane1 or plane2.

Table C.2.1F: Plane Name		
Name	Vertical coordinate	Description
SURF∆∆	Any	surface (*)
ECTOP∆		echo top of radar
Number	**PP**	Pressure [hPa]
	Others	Level index

(*) The sea level pressure is represented as element "PSEA " at plane "SURF ", not "PRES" at the mean sea level.

C.2.2 *Packing Type*

These symbols are used for packing specification of NUSDAS_PARAMETER_CHANGE.

Table C.2.2 Packing Type		
Symbol	deffile	Description
N_P_1PAC	1PAC	packing to signed 8bit integer
N_P_2PAC	2PAC	packing to signed 16bit integer
"2UPC"	2UPC	packing to unsigned 16bit integer
N_P_4PAC	4PAC	packing to signed 32bit integer
N_P_I1	I1	signed 8bit integer
N_P_I2	I2	signed 16bit integer
N_P_N1I2	N1I2	packing to signed 16bit integer: factor is 10
N_P_I4	I4	signed 32bit integer
N_P_R4	R4	IEEE 32bit float
N_P_R8	R8	IEEE 64bit float
N_P_RLEN	RLEN	Run Length Encoding
N_P_GRIB	GRIB	GRIB version 2 (not implemented yet)

C.2.3 *Error Code of NuSDaS library*

All NuSDaS functions in C interface return int, and subroutines in Fortran interface return INTEGER through the last argument. Meaning of the result code is shown in following table. Please note that the meaning of result code larger than -10 depends on what function you called.

Table C.2.3 Common Error Code	
code	Description
-10	memory allocation failure
-11	bad type1 or type2
-12	invalid character in type1/type2/type3
-13	non-existent type1/type2/type3
-19	data type1/type2 completion table does not found
-20	too many definition files are opened
-21	definition file not found nor readable
-31	too large dataset declared by definition file
-32	Memory allocation failure
-33	lines in definition file are missing or not in correct order
-34	malformed ELEMENTMAP specification
-40	type1 undefined in definition file
-41	type2 undefined in definition file
-42	type3 undefined in definition file
-43	number of valid times undefined in definition file
-44	list of valid times undefined in definition file
-45	Number of planes undefined in definition file
-46	list of planes undefined in definition file
-47	number of elements undefined in definition file

-48	list of elements undefined in definition file
-49	size of grid undefined in definition file
-50	invalid base time and valid time combination
-51	data file directory not found or mkdir error
-52	too many data files are opened
-53	cannot create data file
-54	NUSD record malformed or unreadable
-55	CNTL record malformed or unreadable
-56	INDX record malformed or unreadable
-57	END record malformed or unreadable
-60	I/O error (in datafile initialization)
-61	Memory allocation error (in datafile initialization)
-62	specified data is not allowed by definition file (in datafile initialization)
-63	record larger than fixed record size (in datafile initialization)
-64	INFO source file specified but not readable
-65	cannot write NUSD record (in closing of datafile)
-66	cannot write INDX record (in closing of datafile)
-67	cannot write END record (in closing of datafile)
-68	write into read-only file
-69	write into dataset with NRD number 50 or more
-70	open error in ES routines
-71	write error in ES routines
-72	read error in ES routines
-73	I/O error in ES routines
-76	invalid unit number is used in ES routines
-77	I/O error in ES routines
-78	the file is not ES.
-79	memory allocation failure in ES routines
-99	I/O error

C.2.4 *User Data Array Type*

Table C.2.4 gives type symbols are used for argument *utype* for NUSDAS_READ and NUSDAS_WRITE subroutines. Missing value symbols are used as *udata* value for NUSDAS_MAKE_MASK.

C.2.5 *NuSDaS Element Table*

- Table C.2.5 shows the list of registered elements.
- GRIB1 field shows corresponding GRIB (edition 1) parameter indicator. This can be used in conversion **to** GRIB 1. Asterisk '*' before the code indicates that the NuSDaS element name is NOT used in case of conversion **from** GRIB 1, generally because the units is different from that in GRIB.
- GRIB parameter indicator more than 127 is local assignment of Japan Meteorological Agency.
- When you are handling data integrated over time (i.e. the time attribute in *type2* is **AV**), multiply [s] to the units in this table.
- Element names beginning with underline character '_' are used when the time attribute of *type2* is **SV**, which is properly used for dataset of snapshot data. They indicate that the element is time integration (or maybe average). Refer "SUBC TDIF record" to obtain information on integration time. In case of such time-integrated data, you have also to multiply s to the units in this table.

Table C.1.5: *proj* parameter values, returned from NUSDAS_GRID			
Projection	*type1* string	*proj* parameter	Remarks
Cylindrical Equidistance	**LL**	**LL**	.
Lambert Conformal	**LM**	**LMN**	Northern hemisphere (*)
		LMS	Southern hemisphere
Polar Stereo	**PS**	**PSN**	Northern hemisphere
		PSS	Southern hemisphere
Mercator	**MR**	**MER**	.
Gaussian grid	**GS**	**GS**	.
Oblique Lambert Conformal	**OL**	**OL**	Specified for RADAR data of Japan area. Some parameters are due to the agreement between the users.
Section of latitude	**XP**	**XP**	.
Section of longitude	**YP**	**YP**	.
"Free Grid"	**FG**	**FG**	.
RADAR site	**RD**	**RD**	Peculiar to each site
Station data	**ST**	**ST**	.

(*) In case of Lambert Conformal or Polar Stereo, the "N" or "S" character is added to the proj parameter. The character is decided with the location of the basepoint, indicated in the definition file. The equator is equated to northern hemisphere.

Table C.2.4: User data array type symbols, see C.2.4 for description.			
Symbol		type name	
Type	missing value	Fortran	C
N_I1	**N_MV_UI1**	BYTE (strongly discouraged)	N_SI1 (signed char)
N_I2	**N_MV_SI2**	INTEGER(2)	N_SI2 (usu. short)
N_I4	**N_MV_SI4**	INTEGER	N_SI4 (usu. int)
N_R4	**N_MV_R4**	REAL	float
N_R8	**N_MV_R8**	DOUBLE PRECISION	double
N_NC	---	Obsolete (binary representation as stored in data file)	

Table C.2.5: NuSDaS Element Table, see C.2.5 for description.			
NuSDaS	units	GRIB1	Description
P	hPa	*1	Pressure
Pres	Pa	1	
PAI	-	*1	Log pressure
PSEA	hPa	*2	Pressure reduced to MSL
Pmsl	Pa	2	
Ptend	Pa.s-1	3	Pressure tendency
pVOR	K.m2.kg-1.s-1	4	Potential vorticity

sarH	m	5	ICAO Standard Atmosphere reference height
PHI	m2.s-2	*6	Geopotential
gpH	m2.s-2	6	
Z	m	*7	Geopotential height
gpH	gpm	7	
gmH	m	8	Geometrical height
sdH	m	9	Standard deviation of height
tOZON	Dobson	10	Total ozone
T	K	11	Temperature
vT	K	12	Virtual temperature
pT	K	13	Potential temperature
papT	K	14	Pseudo-adiabatic potential temperature
maxT	K	15	Maximum temperature
minT	K	16	Minimum temperature
dT	K	17	Dew-point temperature
TTD	K	18	Dew-point depression (or deficit)
TRate	K.m-1	19	Lapse rate
VIS	m	20	Visibility
Radr1	-	21	Radar spectra (1)
Radr2	-	22	Radar spectra (2)
Radr3	-	23	Radar spectra (3)
PLI50	K	24	Parcel lifted index (to 500 hPa)
Tano	K	25	Temperature anomaly
Pano	Pa	26	Pressure anomaly
gpHan	gpm	27	Geopotential height anomaly
Wave1	-	28	Wave spectra (1)
Wave2	-	29	Wave spectra (2)
Wave3	-	30	Wave spectra (3)
WindD	Degree_true	31	Wind direction
WindS	m.s-1	32	Wind speed
U	m.s-1	33	u-component of wind [X direction]
WindX	m.s-1	*33	
UU	m.s-1	33	u-component of wind [eastward]
V	m.s-1	34	v-component of wind [Y direction]
WindY	m.s-1	*34	
VV	m.s-1	34	v-component of wind [northward]
PSI	m2.s-1	35	Stream function
CHI	m2.s-1	36	Velocity potential
mPSI	m2.s-2	37	Montgomery stream function
sVV	s-1	38	Sigma coordinate vertical velocity
OMG	hPa.h-1	*39	Vertical velocity
VVPa	Pa.s-1	39	

VVm	m.s-1	40	Vertical velocity
aVOR	s-1	41	Absolute vorticity
aDIV	s-1	42	Absolute divergence
VOR	10-6.s-1	*43	Relative vorticity
rVOR	s-1	43	
DIV	10-6.s-1	*44	Relative divergence
rDIV	s-1	44	
vUS	s-1	45	Vertical u-component shear
vVS	s-1	46	Vertical v-component shear
CrntD	Degree_true	47	Direction of current
CrntS	m.s-1	48	Speed of current
CrntU	m.s-1	49	u-component of current
CrntV	m.s-1	50	v-component of current
Q	kg.kg-1	51	Specific humidity
RH	%	52	Relative humidity
HMR	kg.kg-1	53	Humidity mixing ratio
TPW	kg.m-2	54	Precipitable water
VP	Pa	55	Vapour pressure
VPVPD	Pa	56	Saturation deficit
Evap	kg.m-2	57	Evaporation
CIC	kg.m-2	58	Cloud ice
RRate	kg.m-2.s-1	59	Precipitation rate
ThndP	%	60	Thunderstorm probability
RAIN	kg.m-2	61	Total precipitation
RR10	0.1 mm.min-1	*61	
RR60	mm.h-1	*61	
RR3H	8 mm.day-1	*61	
RR6H	4 mm.day-1	*61	
RR1D	mm.day-1	*61	
RR1M	mm.mon-1	*61	
RRfr0	mm	*61	Precipitation from last hour 00min
RRL	kg.m-2	62	Large scale precipitation
RRLpD	mm.day-1	*62	
RRC	kg.m-2	63	Convective precipitation
RRCpD	mm.day-1	*63	
SnRWe	kg.m-2.s-1	64	Snawfall rate water equivalent
SnWe	kg.m-2	65	Water equivalent of accumulated snow depth
SnowD	m	66	Snow depth
MLD	m	67	Mixed layer depth
tTcD	m	68	Transient thermocline depth
mTcD	m	69	Main thermocline depth
mTcan	m	70	Main thermocline anomaly

CLA	%	71	Total cloud cover
CLC	%	72	Convective cloud cover
CLL	%	73	Low cloud cover
CLM	%	74	Medium cloud cover
CLH	%	75	High cloud cover
CWC	kg.m-2	76	Cloud water
TCWC	kg.m-2	*76	
BLI50	K	77	Best lifted index (to 500 hPa)
SnC	kg.m-2	78	Convective snow
SnL	kg.m-2	79	Large scale snow
WatrT	K	80	Water temperature
SST	K	*80	Water temperature [sea surface]
Land	Proportion	81	Land cover (1 = land, 0 = sea)
Sldev	m	82	Deviation of sea level from mean
Z0	m	83	Surface roughness
Albed	%	84	Albedo
SoilT	K	85	Soil temperature
SoilW	kg.m-2	86	Soil moisture content
Veget	%	87	Vegetation
Sali	kg.kg-1	88	Salinity
Dens	kg.m-3	89	Density
Runof	kg.m-2	90	Water run-off
ROF	mm.day-1	*90	
ROFS		*90	Water run-off [surface]
ROFD		*90	Water run-off [gravity drag]
IceC	Proportion	91	Ice cover (1 = ice, 0 = no ice)
ICE			
IceD	m	92	Ice thickness
IceMD	Degree true	93	Direction of ice drift
IceMS	m.s-1	94	Speed of ice drift
IceMU	m.s-1	95	u-component of ice drift
IceMV	m.s-1	96	v-component of ice drift
IceGR	m.s-1	97	Ice growth rate
IceDV	s-1	98	Ice divergence
SNMlt	kg.m-2	99	Snow melt
CWSSH	m	100	Significant height of combined wind waves and swell
WWvD	Degree true	101	Direction of wind waves
WWvSH	m	102	Significant height of wind waves
WWvMP	s	103	Mean period of wind waves
SwvD	Degree true	104	Direction of swell waves
SwvSH	m	105	Significant height of swell waves
SwvMP	s	106	Mean period of swell waves

PWvD	Degree ture	107	Primary wave direction
PWvMP	s	108	Primary wave mean period
2WvD	Degree ture	109	Secondary wave direction
2WvMP	s	110	Secondary wave mean period
RSNB	W.m-2	111	Net short-wave radiation flux (surface)
RLNB	W.m-2	112	Net long-wave radiation flux (surface)
RSNT	W.m-2	113	Net short-wave radiation flux (top of atmosphere)
RLNT	W.m-2	114	Net long-wave radiation flux (top of atmosphere)
RL	W.m-2	115	Long-wave radiation flux
RLUB		*115	ibid. [upward, surface]
RLDB		*115	ibid. [downward, surface]
RLUT		*115	ibid. [upward, top of atm]
RLDT		*115	ibid. [downward, top of atm]
RLUTc		*115	ibid. [upward, top of atm, clear sky]
RLDBc		*115	ibid. [downward, surface, clear sky]
RS	W.m-2	116	Short-wave radiation flux
RSUB		*116	ibid. [upward, surface]
RSDB		*116	ibid. [downward, surface]
RSUT		*116	ibid. [upward, top of atm]
RSDT		*116	ibid. [downward, top of atm]
RSUBc		*116	ibid. [upward, surface, clear sky]
RSDBc		*116	ibid. [downward, surface, clear sky]
RSUTc		*116	ibid. [upward, top of atm, clear sky]
RSDSn		*116	ibid. [downward, in accumulated snow]
GlRad	W.m-2	117	Global radiation flux
BrT	K	118	Brightness temperature
WNRad	W.m-1.sr-1	119	Radiance (with respect to wave number)
WLRad	W.m-3.sr-1	120	Radiance (with respect to wave length)
FLLH	W.m-2	121	Latent heat flux
FLSH	W.m-2	122	Sensible heat flux
BLDsp	W.m-2	123	Boundary layer dissipation
FLMU	N.m-2	124	Momentum flux, u-component
FLMV	N.m-2	125	Momentum flux, v-component
WMixE	J	126	Wind mixing energy
Image	Brightness Level	127	Image data
WatrT	K	128	Water Temperature
CLC2	%	129	Cloud cover
AvTBB	K	130	Averaged blackbody temperature
MnTBB	K	131	Minimum blackbody temperature
SDTBB	K	132	Standard deviation of blackbody temperature
SNCov	%	133	Snow cover
Tsun	J.m-2	134	Global solar irradiation

HZanP	-	140	Probability of high geopotential height anomaly
PSprd	-	141	Ensemble spread of pressure
ZSprd	-	142	Ensemble spread of geopotential height
TSprd	-	143	Ensemble spread of temperature
EAvSLP	Pa	200	Ensemble mean sea-level pressue
EAvZ	gpm	201	Ensemble mean geopotential height
EAvT	K	202	Ensemble mean temperature
EAvU	m.s-1	203	Ensemble mean u-component of wind
EAvV	m.s-1	204	Ensemble mean v-component of wind
ESDSLP	Pa	210	Ensemble standard deviation of pressure
ESDZ	gpm	211	Ensemble standard deviation of geopotential height
ESDT	K	212	Ensemble standard deviation of temperature
ESDU	m.s-1	213	Ensemble standard deviation of u-component of wind
ESDV	m.s-1	214	Ensemble standard deviation of v-component of wind
FGSU	N.m-2		x-component of momentum flux due to short gravity wave
FGSV	N.m-2		y-component of momentum flux due to short gravity wave
FGLU	N.m-2		x-component of momentum flux due to long gravity wave
FGLV	N.m-2		y-component of momentum flux due to long gravity wave
LTRS	W.m-2.s-1		Evaporation
LINT	W.m-2.s-1		Interception
MSC	m		Moisture in canopy
MSG	m		Moisture in ground or grass
TSC	K		Temperature in canopy
TSG	K		Temperature in ground or grass
ISC	m		Ice or frost on canopy
ISG	m		Frost on grass
SoilI	?		Soil ice content
SoilQ	?		Converged heat in soil
TSN	K		Temperature of accumulated snow surface
SnTmp	K		Temperature in accumulated snow
SnQ	?		Converged heat in accumulated snow
SnW	?		Moisture in accumulated snow
SnDen	?		Density of accumulated snow
SnFr	Proportion	*133	Snow cover
KIND	-		Land surface status code
U1	m.s-1	*33	x-component of wind at the lowest model level
V1	m.s-1	*34	y-component of wind at the lowest model level
T1	K	*11	Temperature at the lowest model level
Q1	kg.kg-1	*51	Specific humidity the lowest model level
WET	-		Wetness
UWV	kg.m-1.s-1		Water vapor flux, x-component
VWV	kg.m-1.s-1		Water vapor flux, y-component

RCST	?		Radiation forcing (short wave, top of atm)
RCSB			Radiation forcing (short wave, surfce)
RCLT			Radiation forcing (long wave, top of atm)
RCLB			Radiation forcing (long wave, surfce)
PBLH	m		Height of planetary boundary layer
CVR	Proportion	*71	Cloud cover
HRRS	K.day-1		Heating rate due to short-wave radiation
HRRL			Heating rate due to long-wave radiation
HRCV			Heating rate due to convection
HRLC			Heating rate due to large-scale condensation
HRVD			Heating rate due to vertical diffusion
HRAD			Heating rate due to adiabatic motion
QRCV	kg.kg-1.day-1		Moistening rate due to convection
QRLC			Moistening rate due to large-scale condensation
QRVD			Moistening rate due to vertical diffusion
QRAD			Moistening rate due to adiabatic motion
URCV	m.s-1.day-1		u-component acceleration due to convection
URLC			u-component acceleration due to large-scale condensation
URVD			u-component acceleration due to vertical diffusion
URAD			u-component acceleration due to adiabatic motion
VRCV	m.s-1.day-1		v-component acceleration due to convection
VRLC			v-component acceleration due to large-scale condensation
VRVD			v-component acceleration due to vertical diffusion
VRAD			v-component acceleration due to adiabatic motion
UMF	kg.m-2.s-1		Upward mass flux
UMB			Upward mass flux at bottom of cloud
CWF	J.kg-1		Cloud work function
MXWIN	m		Height of maximum wind speed
TROP1			Height of first tropopause
TROP2			Height of second tropopause
CBTOP			Height of top of cumulonimbus
NUM	-		Station ID number
LAT	degree		Latitude
MLAT	degree		
LON	degree		Longitude
MLON	degree		
HIGH	m		Height
AQC	-		AQC code in AMeDAS
Sunsh	s		Duration of sunshine
SSfr0	min		Duration of sunshine
SEC	s		Time
CSEC			

TDDKK	-		Thunder multiplicity and discharge code
TEC	kA		Thunder current
SM	-		Map factor
PI10LV	-		Radar 10 minutes precipitation index
RR60LV	-		Radar 60 minutes precipitation index
HIGHLV	-		Radar echo top height index