DAO Office Note 95-01

Office Note Series on Global Modeling and Data Assimilation

Richard B. Rood, Head
Data Assimilation Office
Goddard Space Flight Center
Greenbelt, Maryland

Documentation of the GEOS/DAS Observation Data Stream (ODS) Version 1.01

Arlindo da Silva*
Christopher Redder†

Data Assimilation Office, Goddard Laboratory for Atmospheres
* *Goddard Space Flight Center, Greenbelt, Maryland*
† *General Sciences Corporation, Laurel, Maryland*

Goddard Space Flight Center
Greenbelt, Maryland 20771
December 1995

Abstract

This document describes the Observation Data Stream (ODS) format used for pre- and post-analysis station data (non-gridded). These HDF compliant files are intended to serve as input as well as output for the Goddard EOS Data Assimilation System (GEOS/DAS). This office note describes the concept of an ODS file and documents high level FORTRAN access software. It is geared to the producers and users of such data sets.

On-line versions of this document are available from

`ftp://dao.gsfc.nasa.gov/pub/office_notes/on9501.ps.Z` (postscript)

`ftp://niteroi.gsfc.nasa.gov/www/on9501/ods.html` (HTML)

Contents

List of Tables

1 Introduction

This Office Note documents the Observation Data Stream (ODS) concept and its FORTRAN implementation using the NetCDF interface to the Hierarchical Data Format (HDF[1]) developed at the National Center for Supercomputer Applications (NCSA). No prior exposure to NetCDF or HDF is assumed, but familiarity with the general structure of Goddard EOS Data Assimilation System (GEOS/DAS) would be helpful.

Traditionally, all observational data received at DAO are first archived with a consistent home-grown format known as UNPACK. UNPACK files hold all observational data we receive, including data that never makes it to the assimilation process. In preparation for assimilation, a pre-processing system referred to as REPACK selects data from UNPACK files, performs a few simple quality checks and corrections (such as the hydrostatic check), producing as output a REPACK file which is read by GEOS/DAS. REPACK files are usually much smaller than UNPACK files as they are tailored for the needs of the assimilation system. Details of the UNPACK/REPACK systems and file formats can be found in Lamich *et al.* (1996).

Useful by-products of the assimilation system are the innovation files (known internally as *del-files*) which contain the difference between the observations and the first guess (a 6 hour forecast) interpolated to the observation location, along with the quality control marks assigned by GEOS/DAS. These innovation files are the primary input for our covariance modeling effort, as well as for the monitoring and validation systems currently in development. The main disadvantages of the current innovation files are: 1) the actual observation values are not stored in the file, only the difference from the first guess, 2) forecast and observation errors used in the assimilation are not recorded, 3) the analyzed value interpolated to observation location is not present, and 4) the file structure is designed around the original upper air analysis system obtained from NCAR in the early 1980's and does not provide a natural framework for the wealth of new data types coming with the EOS era. Furthermore, an extended innovation file including the observation value and the interpolated analysis will provided a very useful tool for DAO's Climate Analysis and Diagnostics (CAD) group.

The main design goals of ODS are therefore:

1. To unify the REPACK and innovation files into a single consistent format. (ODS files are not meant to be a replacement for UNPACK files.) Because HDF has been adopted by EOSDIS, ODS files should be HDF compliant.

2. Because ODS are to be used for many purposes its structure should be simple, with easy to use access software available. Using HDF also ensures portability of the software and data files. Access software from Matlab and IDL should also be developed, but these are not yet available as of this writing.

3. ODS files produced by the REPACK pre-processing system should include not only data to be directly used by the GEOS/DAS, but also several *passive* data types which are nonetheless useful for validation, tuning or diagnostic studies. Typical examples are significant level data which are not used by GEOS but provide an excellent validation tool. In addition, any new data type should first go through GEOS/DAS as a passive data type, so that bias corrections and random error characterization can be developed off-line based on ODS files.

[1]The conventions adopted in ODS do not strictly adhere to the EOSDIS standard for HDF files.

4. ODS files should be sufficiently self-describing, and include several additional pieces of information about the observation such as sounding identifiers and time stamp.

5. With the increasing sophistication of the assimilation methodology, additional information about the data type (*metadata*) will undoubtly be required. Because this *metadata* can vary widely from data type to data type — and it is of concern only for the assimilation specialist — ODS files should include a minimum of metadata information. Metadata information should be provided in companion files tailored to the specific requirements of each data type. The ODS file should only provide the necessary pointer for the separate metadata file. Guidelines for the preparation of metadata files will be presented elsewhere.

6. The ODS file produced by the REPACK pre-processing system will be termed *pre-analysis* ODS file, and does not contain those attributes which can only be provided by the assimilation system (e.g., the 6 hour forecast, the interpolated analysis value). A *post-analysis* ODS file should be produced by GEOS/DAS, having the same information in the pre-analysis ODS files complemented by the assimilation specific information. Nevertheless, a post-analysis ODS file should also be a valid input for GEOS/DAS as it provides the same information as the pre-analysis ODS file. As pointed out by David Lamich, GEOS/DAS could be configured to read data from a post-analysis ODS file and perform an assimilation using the same quality-control marks of the original assimilation. This capability is extremely useful to isolate the effects of the quality control subsystem when testing the impact of modifications to the system.

The organization of this document is as follows. In section 2 we describe the contents of a pre-analysis ODS file, focusing on the definition of some key data attributes. The additional information in a post-analysis ODS file is discussed in section 3. A set of FORTRAN subroutines to read/write ODS files is summarized in section 4, followed by concluding remarks in section 5. The appendices give the full current list of GEOS/DAS data sources, the detailed description of ODS FORTRAN access software, and discuss the ODS file structure from the NetCDF interface perspective.

2 Pre-analysis ODS files

One self-describing feature of every ODS file is the list of data types (kt) and data sources (kx) in use by GEOS/DAS at the time of file creation. Each observation in the ODS file has a data type/data source index associated with it.

2.1 Data type index (kt) tables

A data type is an integer which identifies the observable. For example, an observation of sea level pressure has $kt = 3$. A complete list of data types currently in use at DAO is given in Table 1. By retrieving the data type information from each ODS file the user will be able to identify each data type and its units.

Table 1: Current list of data types used in GEOS/DAS (kt index). A current version of this list must be encoded in each ODS file for completeness.

kt	*variable*	*units*	*Description*
1	u_s	m/sec	Surface (10m) zonal wind
2	v_s	m/sec	Surface (10m) meridional wind
3	slp	hPa	Sea level pressure
4	u	m/sec	Upper-air zonal wind
5	v	m/sec	Upper-air meridional wind
6	h	m	Upper-air geopotential height
7	w	g/kg	Upper-air water vapor mixing ratio
8	T	Kelvin	Upper-air temperature
9	T_d	Kelvin	Upper-air dew-point temperature
10	rh	%	Upper-air relative humidity
11	q	g/kg	Upper-air specific humidity
12	U_s	m/sec	Surface (10m) wind speed
13	T_s	Kelvin	Surface (10m) temperature
14	T_{ds}	Kelvin	Surface (10m) dew-point temperature
15	rh_s	%	Surface (10m) relative humidity
16	q_s	g/kg	Surface (10m) specific humidity
17	P	mm/day	Precipitation rate
18	Q	mm	Total precipitable water
19	Q_l	mm	Total cloud liquid water
20	c	percent	Fractional cloud cover
21	wx	none	Present weather code

2.2 Data source index (kx) tables

A data source index (kx) is an integer which identifies the origin of the measurement (see Table 2). For example, a measurement with $kx = 7$ was made by a radiosonde. The observation attribute km (metadata index) described below can be used to identify the particular station that provided the report. A complete list of data sources currently used in GEOS/DAS can be found in Appendix A.

2.3 Observation attributes

Besides its value, each observation must be accompanied by a set of attributes describing the measurement[2]. A list of these attributes which must be present in a pre-analysis ODS file is given in Table 3. Notice that all attributes must be assigned, *missing values* are not allowed in an ODS file.

Each data attribute is further described below.

[2]Technically we should refer to some of these attributes as *metadata*. However, we shall reserve the term *metadata* in this document for additional information not present in the ODS files. An example of *metadata* could be the error covariance of a given satellite retrieval.

Table 2: Partial list of data sources used in GEOS/DAS (kx index). A current version of this list must be encoded in each ODS file for completeness. See Appendix A for a complete list.

kx	*Description*
1	Surface Land Obs - 1
2	Surface Land Obs - 2
3	Surface Ship Obs - 1
4	Surface Ship Obs - 2
5	Environment Buoy
6	Drifting Buoy
7	Rawinsonde
8	Pilot Wind
9	Ship Released Rawinsonde
10	Dropwinsonde
	etc.

Data type/source indices (kt/kx). These have already been described above.

Sounding index (ks). This index is needed to allow the assembly of profiles coming from the same sounding during a given synoptic time. For example, a radiosonde reporting profiles of winds, temperature and humidity should have the same sounding index associated with each piece of data. In the case of satellite retrievals, each sounding should have their own index ks, although the data is coming from the same *instrument.* The concept of sounding is extended here to include also radiance measurements: satellites measured radiances at several channels (loosely equivalent to vertical levels) for a given horizontal location. Notice that sounding indices only refer to a given synoptic time. For example, if at 0Z the sounding from station LAX was assigned the index 5217, at the next synoptic time a sounding from the same station could have $ks = 27$. Sounding indices are not tied to station ID's. As currently implemented, there is maximum of 65,535 ($2^{16} - 1$) soundings allowed per synoptic time.

Metadata index (km). This index is intended to point to metadata information stored in a separate metadata file tailored to the particular data source. For example, one could maintain a radiosonde master file which tabulates instrumentation, black-list and bias correction information for each reporting radiosonde. In this case, the metadata information could point to the record in this master file corresponding to the reporting station. In the case of satellite data, the metadata index could point to a record on a separate metadata file which contains the state dependent error covariance for a given sounding. As for sounding indices, metadata indices are not necessarily tied to station ID's. As currently implemented, there is a maximum of 2,147,483,647 ($2^{31} - 1$) metadata indices allowed per synoptic time. If there is no metadata for a given data type, then the value of 0 should be used.

Latitude (lat). The latitude of the observation, in degrees, negative in the southern hemisphere, positive in the northern hemisphere. The latitude is encoded internally with 2 bytes at a resolution of $\approx 0.003°$.

Longitude (lon). The latitude of the observation, in degrees, from 180W to 180E. West longitudes should be encoded as negative numbers, *e.g.*, 90W = -90, 90E = +90. The longitude is encoded internally with 2 bytes at a resolution of $\approx 0.005°$.

Table 3: List of mandatory data attributes for a pre-analysis ODS file. The FORTRAN interface refers to how this attribute is written/read to file using the software described in section 4.

Variable name	*Description*	*FORTRAN Interface*	*Units*	*Valid Range*	*Internal Storage (bytes)*
kt	Data type index	Integer		$[1, 255]$	1
kx	Data source index	Integer		$[1, 65535]$	2
ks	Sounding index	Integer		$[1, 65535]$	2
km	Metadata index	Integer		$[0, 2^{31} - 1]$	4
lat	Latitude	Real	degrees north	$[-90, +90]$	2
lon	Longitude	Real	degrees east	$[-180, +180]$	2
level	Level or channel	Real	hPa/none		4
julian	Julian day	Integer			1
time	Time stamp	Integer	minutes	$[0, 1439]$	2
obs	Observation value	Real	depends on *kt*		4
qc_flag	Quality control flag	Integer		$[0, 65534]$	2
mod_flag	Modification flag	Integer		$[0, 255]$	1

Level. In the case of conventional observations such as temperature soundings, this attribute refers to the vertical level, say 500 hPa. The concept of level is extended here to include the channels of radiance measurements. To allow greater flexibility, the level is recorded internally as a 4 byte float variable. For surface observations, the station pressure should be used for `level` whenever available. If not available, a pressure value consistent with the station altitude and a standard atmosphere should be used. For ships, buoys and other sea platforms the sea level pressure could be used to specify `level`.

Julian day (*julian*). The Julian day of the observations at the time of measurement. Notice that there is a distinction between the *Julian day of the observations* described here, and the *synoptic Julian day* introduced in section 4 and Appendix B. For example, a batch of data corresponding to synoptic time 0Z will contain observations that were measured in the previous day, between 23Z and midnight; in this case the Julian day of the observations could be 1 day less than the synoptic Julian day. The definition of Julian adopted here follows the Numerical Recipes (Press *et al.* 1992). In this convention, the May 23, 1968 corresponds to Julian day 2,440,000. There is a maximum of 255 Julian days allowed on a given ODS file. If the Julian day of the observations is not available, then the synoptic Julian day should be specified.

Time stamp (*time*). This attribute gives the elapsed time in minutes since 0 GMT. It should refer to the actual time of measurement and not to the synoptic time. If the actual time of measurement is not available, then elapsed time since 0GMT corresponding to the synoptic time should be used.

Observation (*obs*). This contains the actual value of the observation. As defined here, the u and v components of a wind field constitute two different observations. Also, a temperature profile at 18 mandatory levels is composed of 18 separate observations. The indices (kt, ks) can be used to group the data as profiles of a given quantity.

Quality control flag (*qc_flag*). This flag contains the result of the several quality control tests performed on the observation. A list of these tests is given in Table 4; see

Pfaendtner *et al.* (1995) and Lamich *et al.* (1996) for additional information on the GEOS-1/DAS quality control and data pre-processing procedures. This flag is currently stored on file using 2 bytes, with bitwise encoding of the result of the several quality control tests performed. For mosts tests, a "0" means that the observation passed the test OR that the test has not been performed; a bit of value "1" means that the observation failed the test. For some tests, *e.g.*, the off-line subjective test, the observation can fail (bit values 11), pass (bit values 00), or be marked as suspicious (bit values 01). The *off-line* tests listed in Table 4 are performed prior to the assimilation cycle, at the "REPACK" level. The *on-line* tests are those performed inside the GEOS Data Assimilation System and for the most part make use of the model first guess (currently, a 6 hour forecast). For a summary of these QC tests see Appendix D.

Table 4: Conventions for the quality control flag (*qc_flag*). See text for details.

Bit Position	*Test Description*	*Possible Values*
1	Off-line gross limit check	0 or 1
2	Off-line climatological check	0 or 1
3	Off-line hydrostatic check	0 or 1
4	Off-line sea level pressure check	0 or 1
5	Off-line integrity test	0 or 1
6	Off-line black-list mark	0 or 1
7–8	Off-line Complex QC test	00 or 01 or 11
9–10	Off-line subjective test	00 or 01 or 11
11	On-line gross limit check	0 or 1
12	On-line *buddy* check	0 or 1
13	On-line final QC decision	0 or 1
14	On-line passive data type mark	0 or 1
15–16	Bits currently not used	00

Modification flag (*mod_flag*). In addition to simple pass/fail tests, both on- and off-line quality control systems can alter the value of an observation to remove known biases, correct for transmission errors, or simply to combine several observations into a single, "average" super-observation. Other observations, such as dew-point temperature, are derived from reported quantities. The *mod_flag* attribute keeps a record of the changes made to the observations during the assimilation process. This flag is stored using 1 byte, with bitwise encoding of the different modifying processes (Table 5).

Table 5: Conventions for the modification flag (*mod_flag*). See text for details.

Bit Position	*Description*	*Possible Values*
1	Observation is a super-ob	0 or 1
2	Observation modified by Complex QC system	0 or 1
3	Bias correction applied	0 or 1
4	Derived data type	0 or 1
5–8	Bits currently not used	0000

2.4 Storage requirements

Excluding overhead, a pre-analysis ODS file requires 27 bytes of storage for each observation. Therefore, a pre-analysis ODS file with an average of 200,000 observations for each of the 4 synoptic times will require $\approx$ 22 Mbytes of storage/day.

3 Post-analysis ODS files

A post-analysis ODS file is an extension of its pre-analysis counterpart. In addition to the attributes listed in Table 3, additional attributes produced by the assimilation system are included (see Table 6). These are further described below.

Table 6: Additional data attributes for a post-analysis ODS file. The FORTRAN interface refers to how this attribute is written/read to file using the software described in section 4.

Variable name	*Description*	*FORTRAN Interface*	*Internal Storage (bytes)*
omf	Observation minus 6h forecast	Real	4
oma	Observation minus analysis	Real	4

Innovation (*omf*). The observation minus the 6 hour forecast used in the statistical analysis interpolated to the observation location. These will be calculated whether the observation was used in the analysis or not. The relevant observation operator will be applied to the forecast state as needed. For example, in the case of radiance observations a radiative transfer model will be applied to the 6 h forecast temperature and moisture profiles to produce the forecast radiance values.

Observation minus analysis (*oma*). The observation minus the analysis interpolated to the observation location. Like the innovation, the relevant observation operator will be applied to the analysis state as needed.

Remark

In no circumstance should a post-analysis ODS file modify the original observations from the pre-analysis ODS file. If super-obing or other device is performed by the assimilation system, a new observation should be created, and the original observation should receive an appropriate quality mark.

3.1 Storage requirements

Excluding overhead, a pre-analysis ODS file requires 35 bytes of storage for each observation. Therefore, a post-analysis ODS file with an average of 200,000 observations for each of the 4 synoptic times will require $\approx$ 28 Mbytes of storage/day.

4 Overview of the FORTRAN interface

This section briefly describes the 8 user callable FORTRAN subroutines which allow users to create, read and write ODS files. Table 7 lists each routine, and Appendix B describes the input/output parameters in some detail. For the sake of completeness, Appendix C describes the structure of the ODS file from the NetCDF interface perspective.

Table 7: User callable FORTRAN subroutines to read/write ODS files. See Appendix B for a complete description of input/output parameters.

Routine Name	*Function*
ODS_Create()	Creates an ODS file
ODS_Open()	Opens a pre-existing ODS file
ODS_PutI()	Writes an INTEGER variable for one synoptic time
ODS_PutR()	Writes an REAL variable for one synoptic time
ODS_GetI()	Reads an INTEGER variable for one synoptic time
ODS_GetR()	Reads a REAL variable for one synoptic time
ODS_Close()	Closes an ODS file
ODS_Append()	Appends data to current synoptic time

Next we outline the main steps involved in writing/reading an ODS file.

4.1 Creating an ODS file

To create a brand new ODS file one needs to have access to a current list of data sources and data types, along with its units. These lists will likely be maintained by DAO's production group. The first step is to call routine `ODS_Create()` to setup the file data structures and save the data type/source lists. You should also provide the first Julian day to be stored on file. The numerical recipes routines `JULDAY` and `CALDAT` can be used to convert from calendar date to Julian day and vice-versa. When creating an ODS file the user must specify whether a *pre-analysis* or *post-analysis* file is desired. On return, `ODS_Create()` returns the integer variable `id` which should be regarded as unit number (or file handle) for subsequent access of the file.

Data is written to an ODS one synoptic time at time. Currently, analyses are performed 4 times a day at synoptic times 0, 6, 12 and 18 GMT. The current practice is that data collected at a windows of $\pm$ 3 hours around the synoptic time are assigned to the synoptic time. (Recall that ODS files now include time stamp information specifying the actual time the measurement was made.) Say it is 12 GMT and you have a total `nobs` observations to be stored on a pre-analysis ODS file. You should then have vectors of length `nobs` with all the relevant data attributes (see section 2). You will need to at least make the following calls after successfully creating the file with `ODS_Create()`:

```
integer  syn_jul, syn_time
integer  kt(nobs), kx(nobs), ks(nobs), km(nobs)
integer  julday(nobs), time(nobs)
integer  qc_flag, mod_flag
```

```
real     lat(nobs), lon(nobs), level(nobs), obs(nobs)

call ODS_PutI ( id, 'kt',       syn_jul, syn_time, nobs, kt,     ierr )
call ODS_PutI ( id, 'kx',       syn_jul, syn_time, nobs, kx,     ierr )
call ODS_PutI ( id, 'ks',       syn_jul, syn_time, nobs, ks,     ierr )
call ODS_PutI ( id, 'km',       syn_jul, syn_time, nobs, km,     ierr )
call ODS_PutI ( id, 'julian',   syn_jul, syn_time, nobs, julday,ierr )
call ODS_PutR ( id, 'lat',      syn_jul, syn_time, nobs, lat,    ierr )
call ODS_PutR ( id, 'lon',      syn_jul, syn_time, nobs, lon,    ierr )
call ODS_PutR ( id, 'level',    syn_jul, syn_time, nobs, level,  ierr )
call ODS_PutI ( id, 'time',     syn_jul, syn_time, nobs, time,   ierr )
call ODS_PutR ( id, 'obs',      syn_jul, syn_time, nobs, obs,    ierr )
call ODS_PutR ( id, 'qc_flag',  syn_jul, syn_time, nobs, obs,    ierr )
call ODS_PutR ( id, 'mod_flag', syn_jul, syn_time, nobs, obs,    ierr )
```

Refer to Appendix B for a description of I/O parameters for `ODS_PutR()`. Notice that the particular order in which the data attributes are written is not relevant. However, if you forget to save a given attribute, undefined values will be stored on the ODS file for that variable. The package performs a test to ensure that the data you are attempting to store are within the ranges specified in Tables 3–6. It is advisable that the **ierr** parameter be checked each time to make sure the data have been properly stored. The possible values of the parameter **ierr** are given in Table 8.

Notice that **syn_jul** and **syn_time** are the *synoptic* Julian day and *synoptic* time of the batch of observations. These are not necessarily the same as the **julian** and **time** attribute listed in Table 3. For example, a batch of data corresponding to synoptic time 0Z will contain observations that were measured in the previous day, between 23Z and midnight.

You can store many synoptic times on a single ODS file, up to a maximum of 255 days. In practice, pre-analysis ODS files will probably have one day worth of data (4 synoptic times), and post-analysis ODS files will depend on the length of the assimilation segment, typically from 1 to 7 days. After all desired synoptic data have been written to the file you should call routine `ODS_Close()` to close the file, providing an **event** string, probably containing the name and version of the program which created the file and the data of creation.

4.2 Reading an ODS file

The first thing to do is to call routine `ODS_Open()` to open the desired file. Usually, one will want to open the file for *reading*, but there is an option for opening the the file for *writing* as well. The `ODS_Open()` also returns the first and last Julian days available on file so that you can make sure you will have the data you want. On return, `ODS_Open()` returns the integer variable **id** which should be regarded as a unit number (or file handle) for subsequent access of the file. Once the file is opened, you can retrieve any data attribute you want, one full synoptic time at a time, with routines `ODS_GetI()` or `ODS_GetR()`, *viz.*

```
integer  syn_jul, syn_time
integer  kt(nobsmx),  kx(nobsmx)
real     lat(nobsmx), lon(nobsmx), level(nobsmx)
real     obs(nobsmx)

call ODS_GetI ( id, 'kt',    syn_jul, syn_time, nobs, kt,    ierr )
```

Table 8: Values of the **ierr** parameter returned by ODS routines described in Appendix B. The positive error codes are issued by the NetCDF routines; negative ones by the ODS package (except for **ierr = -1** which is issued by the NetCDF package). For more details consult the NetCDF manual (Rew *et al.* 1993

ierr	*Description*
0	No Error
1	Not a netcdf id
2	Too many netcdfs open
3	netcdf file exists but user selected no clobber
4	Invalid Argument
5	Write to read only
6	Operation not allowed in data mode
7	Operation not allowed in define mode
8	Coordinates out of Domain
9	MAXNCDIMS exceeded
10	String match to name in use
11	Attribute not found
12	MAXNCATTRS exceeded
13	Not a netcdf data type
14	Invalid dimension id
15	NCUNLIMITED in the wrong index
16	MAXNCVARS exceeded
17	Variable not found
18	Action prohibited on NCGLOBAL varid
19	Not a netcdf file
-1	System error
-3	Invalid dimension size
-4	Invalid interface for variable

```
call ODS_GetI ( id, 'kx',    syn_jul, syn_time, nobs, kx,    ierr )
call ODS_GetR ( id, 'lat',   syn_jul, syn_time, nobs, lat,   ierr )
call ODS_GetR ( id, 'lon',   syn_jul, syn_time, nobs, lon,   ierr )
call ODS_GetR ( id, 'level', syn_jul, syn_time, nobs, level, ierr )
call ODS_GetR ( id, 'obs',   syn_jul, syn_time, nobs, obs,   ierr )
```

Always remember to check parameter **ierr** to make sure the data have been properly read. The possible values of the parameter **ierr** are given in Table 8. As usual, use routine **ODS_Close()** to close the file when you are done. No event tag is necessary when you open the file for reading only.

4.3 Compiling an ODS application program

The first thing that you have to make sure is that HDF is installed in your system. In particular, the NetCDF interface (named MFHDF in the NCSA distribution) which is usually not installed by default. The public domain HDF software can be obtained by

anonymous ftp from `ftp.ncsa.uiuc.edu`, `cd HDF`. NCSA usually provides pre-compiled libraries for most Unix workstations and the Cray supercomputer.

The ODS package is distributed as a FORTRAN 77 source file `ods_hdf.f` and 3 companion include (header) files: `ods_hdf.h`, `ods_worksp.h` and `stdio.h`[3]. If your application program is in a file named `my_program.f`, then the likely Unix command to compile and link this program is

```
% f77 -o my_program.x my_program.f ods_hdf.f -lnetcdf -ldf
```

If your system has the `netcdf` library but not the `df` library this is probably because you have Unidata's implementation of NetCDF. You program will compile allright, but you will not be able to read the standard DAO files produced at DAO.

Please notice that ODS files are written using the NetCDF Application Program Interface (API) but relies on the HDF library for low level I/O. (In the HDF distribution this software is referred to as MFHDF.) *Therefore, a standard ODS file produced at DAO cannot be read by a program linked with Unidata's NetCDF library.*

4.4 Extending variable ranges

To conserve storage, the current implementation places some limits on the amount of information that can be saved on a given ODS file. For example, it is assumed that at most 65,535 ($2^{16} - 1$) soundings can be present in the file. If the need arises in the future, this assumption could be removed without any changes to the FORTRAN interface, and in upwardly compatible manner. Say, we need to increase the range of the sounding index ks. Then, we would need to alter the software so that new files store ks using 4 bytes rather than the 2 bytes as currently implemented. To preserve compatibility with the original ODS files, when opening an ODS file the access software would use NetCDF routine NCVINQ to inquire the type of the variable ks (i.e., 2 byte `short` or 4 byte `long`) and take the appropriate action when reading ks from the file. All these changes should occur internally in the access software without any changes in the calling interface or user program.

5 Concluding remarks

We have discussed the concept of an Observation Data Stream (ODS) for use in the operational GEOS/DAS, as well as for validation, monitoring and data based diagnostic studies. A simple to use FORTRAN access software was described with many of the details of implementation purposely omitted. ODS access software for Matlab and IDL is planed for the near future. Guidelines for companion metadata files will be presented elsewhere.

[3] At DAO, the software will be maintained on hera.gsfc.nasa.gov, `/production/ods`. Consult file **README** in this directory for up-to-date information. Pre-compiled ODS libraries for several popular workstations will also be maintained in this directory.

6 Acknowledgments

A formal technical review of this document was conducted on January 24, 1996 at the Data Assimilation Office. We would like to thank David Lamich (review leader), Yelena Kondratyeva, Jing Guo and John Wu (reviewers) for valuable suggestions. The quality control and modification flag conventions were designed with input from David Lamich, Yelena Kondratyeva, Dick Dee and Alice Trenholme.

Appendix A: Current list of GEOS/DAS Data Sources

kx	*Description*
1	Surface Land Obs - 1
2	Surface Land Obs - 2
3	Surface Ship Obs - 1
4	Surface Ship Obs - 2
5	Environment Buoy
6	Drifting Buoy
7	Rawinsonde
8	Pilot Wind
9	Ship Released Rawinsonde
10	Dropwinsonde
11	Radar-tracked Rawinsonde
12	Rocketsonde
13	Balloon
14	Aircraft - Air/Sat Relay
15	Aircraft - Int. Data Sys
16	Aircraft Report
17	Aircraft Coded Report
18	Aircraft - ALPX
19	Cld Trk Wind - Wisc E1
20	Cld Trk Wind - Wisc E2
21	Cld Trk Wind - Wisc W
22	Cld Trk Wind - Wisc Ocean
23	Cld Trk Wind - Repr. Jap.
24	Cld Trk Wind - NESS East
25	Cld Trk Wind - NESS West
26	Cld Trk Wind - Eurpoean
27	Cld Trk Wind - Japanese
28	SEASAT - Scatterometer
29	WSAT 55
30	WSAT 57
31	Limb Infrared Mon.- Strat
32	User-defined instrument 1
33	NESDIS NH Land AM type A
34	NESDIS SH Land AM type A
35	NESDIS NH Land AM type B
36	NESDIS SH Land AM type B
37	NESDIS NH Land AM type C
38	NESDIS SH Land AM type C
39	NESDIS NH Ocn AM type A
40	NESDIS SH Ocn AM type A
41	NESDIS NH Ocn AM type B
42	NESDIS SH Ocn AM type B
43	NESDIS NH Ocn AM type C
44	NESDIS SH Ocn AM type C
45	NESDIS NH Land PM type A

kx	*Description*
46	NESDIS SH Land PM type A
47	NESDIS NH Land PM type B
48	NESDIS SH Land PM type B
49	NESDIS NH Land PM type C
50	NESDIS SH Land PM type C
51	NESDIS NH Ocn PM type A
52	NESDIS SH Ocn PM type A
53	NESDIS NH Ocn PM type B
54	NESDIS SH Ocn PM type B
55	NESDIS NH Ocn PM type C
56	NESDIS SH Ocn PM type C
57	Special Sat NH - Ocn A
58	Special Sat SH - Ocn A
59	Special Sat NH - Ocn B
60	Special Sat SH - Ocn B
61	Special Sat NH - Ocn C
62	Special Sat SH - Ocn C
63	VAS NH Land - type A
64	VAS SH Land - type A
65	VAS NH Land - type B
66	VAS SH Land - type B
67	VAS NH Ocean- type A
68	VAS SH Ocean- type A
69	VAS NH Ocean- type B
70	VAS SH Ocean- type B
71	NASA-GLA NH Land type A
72	NASA-GLA SH Land type A
73	NASA-GLA NH Land type B
74	NASA-GLA SH Land type B
75	NASA-GLA NH Land type C
76	NASA-GLA SH Land type C
77	NASA-GLA NH Land type D
78	NASA-GLA SH Land type D
79	NASA-GLA NH Ocean type A
80	NASA-GLA SH Ocean type A
81	NASA-GLA NH Ocean type B
82	NASA-GLA SH Ocean type B
83	NASA-GLA NH Ocean type C
84	NASA-GLA SH Ocean type C
85	NASA-GLA NH Ocean type D
86	NASA-GLA SH Ocean type D
87	Pseudo-1000mb Heights
88	ER-2 Aircraft / MMS Data
89	Aircraft reports (ACARS)
90	User-defined instrument 3

kx	*Description*
91	UARS / MLS 1
92	UARS / MLS 2
93	UARS - 3
94	UARS - 4
95	User-defined instrument 4
96	SSM/I - 1
97	SSM/I - 2
98	AMSU NH Land type A
99	AMSU SH Land type A
100	AMSU NH Land type B
101	AMSU SH Land type B
102	AMSU NH Land type C
103	AMSU SH Land type C
104	AMSU NH Land type D
105	AMSU SH Land type D
106	AMSU NH Ocean type A
107	AMSU SH Ocean type A
108	AMSU NH Ocean type B
109	AMSU SH Ocean type B
110	AMSU NH Ocean type C
111	AMSU SH Ocean type C
112	AMSU NH Ocean type D
113	AMSU SH Ocean type D

Appendix B: Observation Data Stream (ODS) user callable routines

B.1 ODS_Create()

Creates an ODS file (including setting up the data structure and contents), sets some internal file parameters and saves text information associated with GEOS/DAS standard data types and data sources.

Two types of ODS can be created:

Pre-analysis: this file contains the observed value, along with space-time coordinates and a metadata indicator. At DAO this file is usually produced by the REPACK pre-processing system.

Post-analysis: In addition to the same information present in the pre-analysis files, additional information produced during the assimilation process is included. Examples are the difference between 6 hour forecast and the observed values, and quality control flags assigned to the observations.

CALLING SEQUENCE:

```
call ODS_Create ( id, filename, ods_type, first_jday,
                  kt_max, kt_names, kt_units,
                  kx_max, kx_names, ierr )
```

INPUT PARAMETERS:

```
character * (*)  filename            ! name of ODS file
character * (*)  ods_type            ! = 'pre_anal', if a pre-analysis
                                     !   ODS file is to be created.
                                     ! = 'post_anal', if a post-analysis
                                     !   ODS file is to be created.
integer          first_jday          ! first julian day to be written
                                     !  to the file.
integer          kt_max              ! maximum number of GEOS/DAS
                                     !   data types
character * (*)  kt_names ( kt_max ) ! names for each data type
character * (*)  kt_units ( kt_max ) ! units for each data type
integer          kx_max              ! maximum number of GEOS/DAS
                                     !   data sources
character * (*)  kx_names ( kx_max ) ! names fore each data source
```

OUTPUT PARAMETERS:

```
integer         id                  ! file handle (or file id);
                                    !  use this to refer to this
                                    !  file later on.
integer         ierr                ! Error code. If non-zero, an
                                    !  error has occurred. For a list
                                    !  of possible values see Table 8
                                    !  of da Silva and Redder (1995).
```

NOTE: No more than id_max (as defined in header file ods_hdf.h) files can be opened at any one time.

SEE ALSO:

ODS_Open()	opens a pre-existing ODS file and returns the dimensions and text data for kt_names, kt_units and kx_names.)
ODS_Close()	closes the ODS file)

REVISION HISTORY:

02Oct95	da Silva	Specification.
13Oct95	Redder	Original code.

B.2 ODS_Open()

Opens an existing ODS file for reading or writing. The Names and units of the standard GEOS/DAS data sources and data types likely to be present in the file are returned.

CALLING SEQUENCE:

```
call ODS_Open  ( id,          filename,    mode,
                 first_jday,  latest_jday, latest_hour
                 kt_max,      kt_names,    kt_units,
                 kx_max,      kx_names,
                 ierr )
```

INPUT PARAMETERS:

```
character * (*)  filename              ! name of ODS file
character * (*)  mode                  ! mode = 'w' open for writing
                                       ! mode = 'r' open for reading
```

OUTPUT PARAMETERS:

```
integer          id                    ! file handle (file id)
                                       !  use this to refer to this
                                       !  file later on.
integer          first_jday            ! first julian day on file
integer          latest_jday           ! latest julian day for which
                                       !   data exists
integer          latest_hour           ! latest julian hour for which
                                       !   data exists
integer          kt_max                ! maximum number of GEOS/DAS
                                       !   data types
character * (*)  kt_names ( kt_max )   ! name for each data type
character * (*)  kt_units ( kt_max )   ! units for each data type
integer          kx_max                ! maximum number of GEOS/DAS
                                       !   data sources
character * (*)  kx_names ( kx_max )   ! name for each data source
integer          ierr                  ! Error code. If non-zero, an
                                       !  error has occurred. For a list
                                       !  of possible values see Table 8
                                       !  of da Silva and Redder (1995).
```

NOTE: No more than id_max (as defined in header file ods_hdf.h)

files can be opened at any one time.

SEE ALSO:

ODS_Create()	creates the ODS file. sets the dimensions and saves the text data for kt_names, kt_units and kx_names.

REVISION HISTORY:

02Oct95	da Silva	Specification.
13Oct95	Redder	Original code.

B.3 ODS_Close()

Closes an ODS file. If the file has been opened for writing a history tag string is saved on the file prior to closing.

CALLING SEQUENCE:

```
call ODS_Close ( id, event, ierr )
```

INPUT PARAMETERS:

```
integer            id        ! file handle as returned from
                               !  ODS_Create() or ODS_Open().
character * ( * )  event     ! program event tag - usually the
                               !  name of the program which modified
                               !  the file and a date.
```

OUTPUT PARAMETERS:

```
integer            ierr      ! Error code. If non-zero, an error
                               !  has occurred. For a list of
                               !  possible values see Table 8
                               !  of da Silva and Redder (1995).
```

SEE ALSO:

ODS_Create()	creates an ODS file
ODS_Open()	opens an existing ODS file
ods_hdf.h	an include file defining internal constants and global variables, and setting up data structures
stdio.h	an include file defining standard input/output unit numbers

SYSTEM ROUTINES:

The NetCDF API is used to access the HDF/ODS file.

REVISION HISTORY:

02Oct95	da Silva	Specification.
16Feb96	Redder	Original code.

B.4 ODS_PutI()

Stores the contents of an **integer** variable for a full synoptic hour on an opened ODS file.

Notes:

1. This routine does perform checks to verify whether the range of values on input is consistent with the internal variable type in the ODS file. For example, a real variable here could be stored as a two byte variable on file. Therefore, all elements in the array to be stored as two byte integers should have values between -32,767 and 32,767. If the software does detect a number inconsistent with the variable type, then the routine returns with an error message without saving any values. This check prevents overflows from occurring which would produce unexpected results and probubly abnormal program termination.

2. The input values should be within the range specified by Table 3 in da Silva and Redder (1996) or by the ODS file annotated header of the ODS file produced with the MFHDF utility, ncdump (see Table 7 in da Silva and Redder, 1996). If no range is specified, then the maximum and minimum values for the internal variable types are as follows:

type	minimum value	maximum value
byte	0	255
short (2 byte) integer	-32,767	32,767
long (4 byte) integer	-2,147,483,643	2,147,483,643
real (4 bytes)	-3.40e38	3.40e38

Reference:

da Silva, A. and C. Redder, 1996: Documentation of the GEOS/DAS Observation Data Stream (ODS) Version 1.01, *DAO Office Note 95-01*, NASA Goddard Space Flight Center, Greenbelt, MD, 30pp.

CALLING SEQUENCE:

```
call ODS_PutI ( id ,  VarName, julian_day, syn_hour,
                nval, values,  ierr )
```

INPUT PARAMETERS:

```
integer         id              ! file handle as returned from
```

```
                                        !  ODS_Create() or ODS_Open().
character * (*) VarName                 ! name of variable
integer         julian_day              ! Julian Day
integer         syn_hour                ! hour of synoptic time since
                                        !   0:00 GMT (e.g., 0, 6, 12,
                                        !   18)
integer         nval                    ! number of values for this
                                        !  synoptic time.
integer         values ( nval )         ! values to be written
```

OUTPUT PARAMETERS:

```
integer          ierr                   ! Error code. If non-zero, an error
                                        !  has occurred. For a list of
                                        !  possible values see Table 8
                                        !  of da Silva and Redder (1995).
```

SEE ALSO:

```
ODS_GetI()    get   a list of integer values from an ODS file
ODS_PutR()    write a list of real    values to   an ODS file
ODS_GetR()    get   a list of real    values from an ODS file
ods_hdf.h     include file defining internal constants
               and global variables, and setting up data
               structures
```

REVISION HISTORY:

```
02Oct95   da Silva    Specification.
16Feb96   Redder      Original code.
```

B.5 ODS_GetI()

Reads the contents of an integer variable for a full synoptic time from an opened ODS file.

CALLING SEQUENCE:

```
call ODS_GetI ( id , VarName, julian_day, syn_hour,
                     nval,    values,     ierr )
```

INPUT PARAMETERS:

```
integer          id              ! file handle as returned from
                                 !   routines ODS_Create()
                                 !   and ODS_Open
character * (*)  VarName         ! name of variable
integer          julian_day      ! Julian Day
integer          syn_hour        ! hour of synoptic time since
                                 !   0:00 GMT (e.g., 0, 6, 12,
                                 !   18)
```

INPUT PARAMETERS:

```
integer          nval            ! on input: maximum number of
                                 !   values (usually determined
                                 !   by the space allocated for
                                 !   storage)
                                 ! on output: the number of
                                 !   values obtained from the
                                 !   file
                                 ! note: If the input value is
                                 !   smaller than the number
                                 !   of values to be read, then
                                 !   the routine exits with a
                                 !   system error message and
                                 !   code of ODS_DimErr ( = -3 ).
                                 !   Also, nval is set to the
                                 !   minimum value required in
                                 !   order for the routine to
                                 !   execute successfully.
```

OUTPUT PARAMETERS:

```
integer         values ( nval ) ! values to be written
integer          ierr           ! Error code. If non-zero, an error
                                !  has occurred. For a list of
                                !  possible values see Table 8
                                !  of da Silva and Redder (1995).
```

SEE ALSO:

ODS_PutI()	write a list of integer values to an ODS file
ODS_PutR()	write a list of real values to an ODS file
ODS_GetR()	get a list of real values from an ODS file
ods_hdf.h	include file defining internal constants and global variables, and setting up data structures

REVISION HISTORY:

02Oct95	da Silva	Specification.
16Feb96	C. Redder	Original code

B.6 ODS_PutR()

Stores the contents of a **real** variable for a full synoptic hour on an opened ODS file.

Notes:

1. This routine does perform checks to verify whether the range of values on input is consistent with the internal variable type in the ODS file. For example, a real variable here could be stored as a two byte variable on file. Therefore, all elements in the array to be stored as two byte integers should have values between -32,767 and 32,767. If the software does detect a number inconsistent with the variable type, then the routine returns with an error message without saving any values. This check prevents overflows from occurring which would produce unexpected results and probubly abnormal program termination.

2. The input values should be within the range specified by Table 3 in da Silva and Redder (1996) or by the ODS file annotated header of the ODS file produced with the MFHDF utility, ncdump (see Table 7 in da Silva and Redder, 1996). If no range is specified, then the maximum and minimum values for the internal variable types are as follows:

type	minimum value	maximum value
byte	0	255
short (2 byte) integer	-32,767	32,767
long (4 byte) integer	-2,147,483,643	2,147,483,643
real (4 bytes)	-3.40e38	3.40e38

Reference:

da Silva, A. and C. Redder, 1996: Documentation of the GEOS/DAS Observation Data Stream (ODS) Version 1.01, *DAO Office Note 95-01*, NASA Goddard Space Flight Center, Greenbelt, MD, 30pp.

CALLING SEQUENCE:

```
call ODS_PutR ( id , VarName, julian_day, syn_hour,
                     nval,    values,     ierr )
```

INPUT PARAMETERS:

```
integer         id                ! file handle as returned from
```

```
                                        ! routines ODS_Create() or
                                        ! ODS_Open().
character * (*) VarName                 ! name of variable
integer         julian_day              ! Julian Day
integer         syn_hour                ! hour of synoptic time since
                                        !   0:00 GMT (e.g., 0, 6, 12,
                                        !   18)
integer         nval                    ! number of values for this
                                        !   synoptic time
real            values ( nval )         ! values to be written
```

OUTPUT PARAMETERS:

```
integer         ierr                    ! Error code. If non-zero, an error
                                        !  has occurred. For a list of
                                        !  possible values see Table 8
                                        !  of da Silva and Redder (1995).
```

SEE ALSO:

```
ODS_PutI()    write a list of integer values to   an ODS file
ODS_GetI()    get   a list of integer values from an ODS file
ODS_GetR()    get   a list of real    values from an ODS file
ods_hdf.h     include file defining internal constants
               and global variables, and setting up data
               structures
```

REVISION HISTORY:

```
02Oct95   da Silva    Specification.
16Feb96   Redder      Original code
```

B.7 ODS_GetR()

Reads the contents of a real variable for a full synoptic time from an opened ODS file.

CALLING SEQUENCE:

```
call ODS_GetR ( id , VarName, julian_day, syn_hour,
                     nval,    values,     ierr )
```

INPUT PARAMETERS:

```
integer          id              ! file handle that is obtained
                                 !   by invoking one of the
                                 !   following subroutines:
                                 !   ODS_Create and ODS_Open
character * (*)  VarName         ! name of variable
integer          julian_day      ! Julian Day
integer          syn_hour        ! hour of synoptic time since
                                 !   0:00 GMT (e.g., 0, 6, 12,
                                 !   18)
```

INPUT PARAMETERS:

```
integer          nval            ! on input: maximum number of
                                 !   values (usually determined
                                 !   by the space allocated for
                                 !   storage)
                                 ! on output: the number of
                                 !   values obtained from the
                                 !   file
                                 ! note: If the input value is
                                 !   smaller than the number
                                 !   of values to be read, then
                                 !   the routine exits with a
                                 !   system error message and
                                 !   code of ODS_DimErr ( = -3 ).
                                 !   Also, nval is set to the
                                 !   minimum value required in
                                 !   order for the routine to
                                 !   execute successfully.
```

OUTPUT PARAMETERS:

```
real          values ( nval ) ! values to be written
integer        ierr           ! Error code. If non-zero, an error
                              !  has occurred. For a list of
                              !  possible values see Table 8
                              !  of da Silva and Redder (1995).
```

SEE ALSO:

```
ODS_PutI()    write a list of integer values to   an ODS file
ODS_GetI()    get   a list of integer values from an ODS file
ODS_PutR()    write a list of real    values to   an ODS file
ods_hdf.h     include file defining internal constants
               and global variables, and setting up data
               structures
```

REVISION HISTORY:

```
02Oct95   da Silva    Specification.
16Feb96   C. Redder   Original code.
```

B.8 ODS_Append

Sets up the software package to add additional observation reports to the data block specified by the current values for the julian day and synoptic time (stored in common).

CALLING SEQUENCE:

```
call ODS_Append ( id, nval, ierr )
```

INPUT PARAMETERS:

```
integer         id              ! file handle as returned by
                                !   routines ODS_Create() and ODS_Open().
                                !
integer         nval            ! the number of values to be
                                !   added.
```

OUTPUT PARAMETERS:

```
integer          ierr           ! Error code. If non-zero, an error
                                !  has occurred. For a list of
                                !  possible values see Table 8
                                !  of da Silva and Redder (1995).
```

SEE ALSO:

```
ODS_PutI()     write a list of integer values to   an ODS file
ODS_GetI()     get   a list of integer values from an ODS file
ODS_PutR()     write a list of real    values to   an ODS file
ODS_GetR()     get   a list of real    values to   an ODS file
ods_hdf.h      include file defining internal constants
                and global variables, and setting up data
                structures
```

REVISION HISTORY:

```
02Oct95    da Silva     Specification.
16Feb96    C. Redder    Original code.
```

Appendix C: ODS file structure

This appendix describes the structure of the ODS file using the NetCDF interface to HDF (MFHDF). Most users will rely on the provided FORTRAN subroutines (section 4) to access the data without the need of knowing the details of the file structure. Such information is important for those users desiring to access the data using the HDF C Language interface or from application programs such as IDL. It is assumed that the reader has familiarity with basic NetCDF concepts. For an introduction to NetCDF consult Rew *et al.* (1993).

Table 9 presents the header of a typical ODS file (in NetCDF jargon Table 9 presents the CDL description of the file.) Most of the variable names and sizes have already been discussed in sections 2—3. Notice that all synoptic times are stored contiguously in a long one-dimensional vector. To gain access to a segment for a particular synoptic time/Julian day, the "pointers" **syn_beg** and **syn_len** can be used. Given a Julian day and a synoptic time, **syn_beg** provides the beginning of the data segment, and **syn_len** how many observations are available for that particular synoptic time.

Table 9: Annotated header of an ODS file produced with MFHDF utility **ncdump**.

```
netcdf ods {
dimensions:
        nobs = UNLIMITED ;
        ktmax = 21 ;
        kxmax = 87 ;
        ndays = 255 ;
        nsyn = 4 ;
        strlen = 50 ;

variables:

// Data type/source tables:

        char kt_names(ktmax, strlen) ;
                kt_names:name = "Name of GEOS/DAS data types" ;
        char kt_units(ktmax, strlen) ;
                kt_units:name = "Units for each GEOS/DAS data type" ;
        char kx_names(kxmax, strlen) ;
                kx_names:name = "Name of GEOS/DAS data sources" ;

// Space-time coordinates:

        short lat(nobs) ;
                lat:name = "Latitude" ;
                lat:units = "degrees north" ;
                lat:valid_range = -90.f, 90.f ;
                lat:scale_factor = 0.0027466659f ;
        short lon(nobs) ;
                lon:name = "Longitude" ;
                lon:units = "degrees east" ;
                lon:value_at_90E = 90.f ;
                lon:value_at_90W = -90.f ;
                lon:valid_range = -180.f, 180.f ;
```

```
            lon:scale_factor = 0.0054933317f ;
        float level(nobs) ;
            level:name = "Pressure level or channel" ;
            level:units = "hPa or none" ;
        short julian(nobs) ;
            julian:name = "Julian day" ;
            julian:reference_date = "1968-05-23" ;
            julian:value_at_reference_date = 2440000 ;
            julian:add_offset = 17999 ;
            julian:valid_range = 17999, 18254 ;
            julian:latest_jday = 18005 ;
        short time(nobs) ;
            time:name = "Time stamp since 0 GMT" ;
            time:units = "minutes" ;
            time:valid_range = 0, 1432 ;
            time:latest_synoptic_hour = 18 ;

// Data indices:

        byte kt(nobs) ;
            kt:name = "Data type index" ;
            kt:add_offset = 1 ;
            kt:valid_range = 1, 256 ;
        short kx(nobs) ;
            kx:name = "Data source index" ;
            kx:add_offset = 32768 ;
            kx:valid_range = 1, 65535 ;
        short ks(nobs) ;
            ks:name = "Sounding index" ;
            ks:add_offset = 32768 ;
            ks:valid_range = 1, 65535 ;
        long km(nobs) ;
            km:name = "Metadata index" ;
            km:add_offset = 32767 ;
            km:valid_range = 0, 65534 ;
            km:missing_value = 0 ;

// Observed values:

        float obs(nobs) ;
            obs:name = "Observation value" ;
            obs:missing_value = 9.9999999e+14f ;


// Assimilation related information:

        float omf(nobs) ;
            omf:name = "observation minus forecast" ;
        float oma(nobs) ;
            oma:name = "observation minus analysis" ;
        byte qc_flag(nobs) ;
            qc_flag:name = "Quality control flag" ;
            qc_flag:scale_factor = 0.0099999998f ;
```

```
// Internal pointers:

        long days(ndays) ;
        long syn_beg(ndays, nsyn) ;
                syn_beg:name = "Begining of synoptic hour for each day" ;
        long syn_len(ndays, nsyn) ;
                syn_len:name = "Number of observations for syn. time" ;

// Global attributes:

                :source = "Data Assimilation Office, Code 910.3, NASA/GSFC" ;
                :title = "GEOS/DAS Observational Data Stream (ODS) File" ;
                :version = "1.01" ;
                :data_info = "Contact data @dao.gsfc.nasa.gov" ;
                :history = "test1$ `` ;
}
```

Appendix D: Summary of quality control tests

This Appendix presents a brief summary of the quality control tests listed in Table 4. For a description of the observational data preparation for GEOS-1/DAS consult Lamich *et al.* (1996). The on-line quality control system used in GEOS-1/DAS is described in Pfaendtner *et al.* (1995).

Off-line tests

These are tests performed prior to the data assimilation cycle at the data pre-processing stage. At DAO this is system is sometimes referred to as the "REPACK".

Off-line gross limit check: the value of the observations is checked for physical plausibility. For example, wind speeds of 1000 m/s would fail this test.

Off-line climatological check: observations are checked against climatological means and standard deviations. A climatological test was not implemented in GEOS-1/DAS.

Off-line hydrostatic check: radiosonde temperature and geopotential heights are checked to ensure a certain degree of hydrostatic balance.

Off-line sea level pressure check: the station elevation is checked, and in case of very high altitudes the reported sea level pressure fails the test. When the station elevation is not available, station pressure is used to flag stations with high altitudes (very low station pressure values).

Off-line integrity test: vertical profiles are checked for unreasonable and abrupt changes. This check is perfomed primarily on satellite retrievals.

Off-line black-list test: a list of radiosondes with a history of poor reporting practice is consulted, and observations from these stations fail this test.

Off-line Complex QC test: this quality mark is derived from the Complex Quality Control system of Gandin (1988) and Collins and Gandin (1990). As a result, an observation can pass or fail the test, or be marked as suspicious.

Off-line subjective test: this is the result of a subjective test by the *on-duty* meteorologist. This quality mark is generally provided by operational Numerical Weather Prediction centers from which some of our data originate. As a result, an observation can pass or fail the test, or be marked as suspicious.

On-line tests

Theses tests are performed during the data assimilation cycle, and for the most part make use of a 6 hour model forecast used as first guess for the analysis.

On-line gross limit check: the observation is checked against the model first guess, and fails the test if it differs from it by a large amount. These trimming limits are a function of the prescribed forecast and observation error characteristics.

On-line buddy check: Observations that failed some of the off-line tests and/or the on-line gross limit check are marked as suspicious, and are then examined against other "good" observations in its surroundings, or *buddies*. If these suspicious observations are consistent with their *buddies* they will pass the buddy test and will be accepted for use in the analysis.

On-line final QC decision: after all is said and done, this flag will indicate whether an observation passed all quality control tests and is ready to be used in the analysis system. Notice that an observation that passed the final QC decision is not garanteed to be used in the analysis. For example, our current Optimal Interpolation scheme employs data selection, and only a small number of observations near a grid-point are used in the analysis.

On-line passive data type mark: prior to implementation, one usually needs to collect statistics about a new data-type. The passive data type mark allows the data to go through the assimilation system without being quality controled or used in the assimilation. However, observation minus forecast, and observation minus analysis residuals are computed and stored in the output post-analysis ODS file.

References

Collins, W. G. and L. S. Gandin, 1990: Comprehensive hydrostatic quality control at the National Meteorological Center. *Mon. Wea. Rev.*, **18**, 2754–2767.

Gandin, L. S., 1988: Complex quality control of meteorological observations. *Mon. Wea. Rev.*, **116**, 1138–1156.

Lamich, D., M. Hall, Y. Kondratyeva, R. Govindaraju and E. Hartnett, 1996: Documentation of Observational Data Preparation for the Goddard Earth Observing System (GEOS) Data Assimilation System. *NASA Tech. Memorandum 104606, in preparation.*

Pfaendtner, J., S. Bloom, D. Lamich, M. Seablom, M. Sienkiewicz, J. Stobie, A. da Silva, 1995: Documentation of the Goddard Earth Observing System (GEOS) Data Assimilation System–Version 1. *NASA Tech. Memo. No. 104606*, Vol. 4, Goddard Space Flight Center, Greenbelt, MD 20771. Available electronically on the World Wide Web as `ftp://dao.gsfc.nasa.gov/pub/tech_memos/volume_4.ps.Z`

Press, W. H., B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, 1992: *The numerical recipes in Fortran*, Second Ed. Cambridge University Press, New York, USA, 963pp.

Rew, R., G. Davis and S. Emmerson, 1993: *NetCDF User's Guide, an Interface for Data Access.* Unidata Program Center, University Corporation for Atmospheric Research, Boulder, CO, 186pp.