(4)

AFGL-TR-89-0078

EVALUATION OF THE NSSDC COMMON DATA FORMAT

C. E. Jordan
H. J. Singer

Radex., Inc.
Three Preston Court
Bedford, MA 01730

March 7, 1989

Scientific Report No. 2

AIR FORCE GEOPHYSICS LABORATORY
AIR FORCE SYSTEMS COMMAND
UNITED STATES AIR FORCE
HANSCOM AIR FORCE BASE, MASSACHUSETTS 01731-5000

"This technical report has been reviewed and is approved for publication"

EDWARD C. ROBINSON
Contract Manager
Data Systems Branch

ROBERT E. McINERNEY
Branch Chief
Data Systems Branch

FOR THE COMMANDER

C. NEALON STARK
Division Director
Aerospace Engineering Division

This report has been reviewed by the ESD Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS).

Qualified requestors may obtain additional copies from the Defense Technical Information Center. All others should apply to the National Technical Information Service.

If your address has changed, or if you wish to be removed from the mailing list, or if the addressee is no longer employed by your organization, please notify AFGL/DAA, Hanscom AFB, MA 01731. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document requires that it be returned.

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE

ADA207821

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution unlimited
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) RX-890301	5. MONITORING ORGANIZATION REPORT NUMBER(S) AFGL-TR-89-0078
6a. NAME OF PERFORMING ORGANIZATION Radex, Inc. — 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Air Force Geophysics Laboratory
6c. ADDRESS (City, State, and ZIP Code) Three Preston Court Bedford, MA 01730	7b. ADDRESS (City, State, and ZIP Code) Hanscom AFB, Massachusetts 01731-5000
8a. NAME OF FUNDING / SPONSORING ORGANIZATION — 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER Contract F19628-87-C-0084
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION NO.
62101F	9993	XX	YN

11. TITLE (Include Security Classification)

Evaluation of the NSSDC Common Data Format

12. PERSONAL AUTHOR(S)

Carolyn Jordan, Howard J. Singer **

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Scientific Report #2	FROM 3/88 TO 3/89	1989 March 07	20

16. SUPPLEMENTARY NOTATION

**AFGL/PHG, Hanscom AFB, MA 01731

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Common Data Format; Flatfiles; NSSDC; Storage and Timing Comparisons. (JES)

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

The National Space Science Data Center has developed a data format system called the Common Data Format (CDF) which is designed to be useful to many different disciplines and flexible enough to accommodate many different types of data sets. The Air Force Geophysics Laboratory, Space Plasmas and Fields Branch has evaluated this format (VAX/MMS Fortran Version 1.0) comparing it to another common format system currently in use, the Flat Data Base Management System (FlatDBMS), and with Fortran direct access files with various record structures. The storage space required by the two packages is comparable to that of the Fortran direct access files and not greatly in excess of that required by the data itself. Of the two packages, CDF is more flexible and is adaptable to a broader range of data sets. However, data retrieval is faster using the FlatDBMS. The CDF requires 2.5 times the CPU time needed by the FlatDBMS to read, smooth, and write a data set to a file structure which was the same as that read in. We understand that this deficiency of the CDF will be

(OVER)

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	Unclassified

22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code)	22c. OFFICE SYMBOL
E. C. Robinson	(617) 377-3840	AFGL/LCY

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

Abstract Continued:

overcome in future versions which will allow random aggregate access of the data. This evaluation of CDFs is based on easily quantified features such as storage space and CPU time. However, one should keep in mind that there are several important attributes of a common data structure which are not so readily measured but contribute significantly to its performance. These features include the self-documenting data structure, the capability for handling diverse multidimensional data, and the advantages for facilitating data exchange and software development. Any decision regarding the implementation of data formats needs to consider both the quantative and qualitative features.

EVALUATION OF THE NSSDC COMMON DATA FORMAT

TABLE OF CONTENTS

1. Introduction

There is interest in the scientific community in adopting a data format which is useful to many disciplines and flexible enough to accommodate many different data sets (flatfiles, multi-dimensional files, images, etc.). There are many advantages of such a system such as easier sharing of data sets and reduced duplication of effort in developing application software. However, as with any higher level system, there is an additional cost due to increased storage requirements and CPU data-access time. Thus, the question becomes one of expense. Does the common format provide sufficient benefits to compensate for the additional resource requirements?

The National Space Science Data Center (NSSDC) has developed a system called Common Data Format (CDF) [Treinish and Gough, 1987]. Since this format has the potential for widespread use in the space science community, the Air Force Geophysics Laboratory, Space Plasmas and Fields Branch, decided to evaluate the CDF package. The CDF system is compared with one of the other systems currently used, the Flat Data Base Management System (FlatDBMS) [Smith and Clauer, 1986a] and with Fortran direct access files with various record structures.

The following sections will summarize key points of the two data format packages, discuss storage requirements of the same data in these different formats versus the Fortran direct access files, and compare the CPU time required to read and write data from each of the files. Finally, some comments will be made regarding the particular strengths and weaknesses of these format options.

2. FlatDBMS Summary

This package was developed at Stanford University [Smith and Clauer, 1986b]. It is a non-hierarchical file structure which uses two files (one containing data, the other is metadata) to store the data in simple tables and record descriptive information about the data set. The data is written to a direct (random) access file with a fixed record length. There are seven basic subroutines which create and access the "flatfile". These are:

FFCREATE -- creates a flatfile,
FFOPEN -- opens an existing flatfile,
FHPUT -- writes information to the header file,
FHGET -- reads information from the header file,
FDPUT -- writes data to the data file,
FDGET -- reads data from the data file,
FFCLOSE -- closes the flatfile.

There are also several utility subroutines which perform specific operations on the flatfiles. These are intended to simplify the writing of application software, rather than affect the data format itself. Plus, there is also a menu-driven interactive interface for "browsing" through the data set.

Even though this is a generalized system, certain assumptions must be made about the files in terms of time representation, array sizes, data types, header file format, and file units. However, most of these assumptions may be changed within the source code of the routines as necessary. The data is assumed to be in time series with a double precision number in the first data column. The data is assumed to be 4-byte real numbers (however, one may use a Fortran EQUIVALENCE statement should 8-bytes be desired; this is what is done for the time data in the first column). There is a limit of 100 data columns. The header records have an 80 character limit. One may have six files open at once. This system is designed to be used with Fortran.

The authors of this package acknowledge some points requiring improvement. Primarily, there is a problem with the efficiency of using sequential data. Reading one record at a time using a binary search to pinpoint the desired record is quite slow. Thus, this system is not ideal for processing large numbers of consecutive records. Also, they note that disk space is used inefficiently for the sake of simplicity. This particular problem is prohibitive should one have a strongly hierarchical data set.

3. CDF Summary

This package allows for the representation of multi-dimensional data sets as well flat data sets [Gough et al., 1988]. It also has a direct access structure. This system may currently be used with both Fortran and C codes. There are thirteen basic subroutines which create and access the CDF. These are:

CDF_create	-- create a new set of CDF files,
CDF_delete	-- delete a set of CDF files,
CDF_open	-- open an existing set of files,
CDF_close	-- close an open set of files,
CDF_inquire	-- obtain information regarding any CDF,
CDF_var_create	-- create a variable,
CDF_var_put	-- write data to the variable file,
CDF_var_get	-- read data from the variable file,
CDF_var_inquire	-- obtain information about the variable,
CDF_attr_create	-- create an attribute,
CDF_attr_put	-- write information to header file,
CDF_attr_get	-- read information from the header file,
CDF_attr_inquire	-- obtain information about the attribute.

The CDF also contains data and metadata, but unlike the flatfile which does this using only two files, the CDF has two files (with extensions .CDF and .CDH) plus a file for every variable (with extensions .Vnn, where nn is the

variable number). The .CDH file contains the header information. The .CDF file contains the format information for the variables themselves. There are six supported data types: BYTE, STRING, INT*2, INT*4, REAL*4, and REAL*8. Variables may be single values or arrays. However, all the variables must have the same dimension. This is so that each variable may be correlated with every other variable. Not every variable requires the full dimensions, thus the user specifies how a given variable changes with each record or with a particular dimension to avoid excess repetition of data values. One may have up to ten dimensions and there is no limit on the dimension size.

This system is compatible with various application software from NSSDC. Thus, this system is also more extensive than just the basic subroutines imply. The authors of this package have three recommendations to improve performance. First, avoid skipping around the file as much as possible. This can lead to excessive paging of information into and out of memory and thereby degrade execution time. Second, multi-dimensional arrays are stored in row major order. Thus, if one is looping through an array, the innermost loop should correspond to the first index and the outermost loop should correspond to the last index. Third, variable numbers rather than names should be used if time is critical.

As with the FlatDBMS package, each variable is accessed individually. This is ideal for random access, however, it is highly inefficient for going through an extensive set of consecutive records. Unlike the FlatDBMS package, the source code to these routines is not available. This has advantages and disadvantages. It is useful in preventing a single user from tailoring the package to the point where it is no longer compatible with the original version thereby defeating the purpose of a generalized format. However, it also makes minor adjustments impossible and greatly hinders debugging application code which uses these routines.

4. Storage Comparisons -- Flatfiles

The first test of the CDF was that of data storage overhead. Does a CDF require an excessive amount of storage space beyond that which is required by the data alone? To begin with, a flatfile comprised of 2068 records with each record containing an 8-byte word for the time variable and 21 4-byte words for the magnetic field values Bx, By, and Bz from seven ground stations was used. The disk space required for the data alone is 372 blocks on the VAX (each block contains 512 bytes).

A Fortran file containing this data and written with an unformatted WRITE statement occupies 388 blocks. The FlatDBMS format requires 372 blocks for the data alone (this does not include header information). Using a flat CDF where each piece of data is assigned a variable (i.e., 22 variables) requires 390 blocks for the data alone. A one-dimensional CDF using only four variables (time, Bx, By, and Bz) each with a dimension of seven used 569 blocks. This was due to the time being written seven times for each record. Clearly, this was unnecessary. It may be prevented by setting the dim_variances parameter equal to false in the application code. Having done this, the four variable CDF only needed 375 blocks. A two-dimensional CDF requiring only two variables (time and B, where B has dimensions of (7,3) for the seven stations and the three components of the field) required only 373 blocks for the data alone.

In this discussion (and in the table below), no account has been taken of the storage requirements for the header information. This is built-in for the Fortran file which is why its overhead is somewhat higher than that of the FlatDBMS file and CDF file. Both of these database systems require header information, but they are somewhat flexible in terms of length. For the CDF, there are two parts to the header

information. The .CDF file serves as the directory which links together all of the files in the CDF. The size of this directory is determined by the number of variables and is not controllable by the user. For the 22 variable set, it requires 11 blocks, for the 4 variable set, it requires 5 blocks, and for the 2 variable set, it requires 4 blocks. In addition to this, there is the .CDH file which contains the metadata about the data set itself. It is this file which is flexible with the user determining its length. For this particular data set, the FlatDBMS file contained seven blocks of header information. The same information in a CDF format occupied six blocks in the .CDH file.

File Type	Storage Blocks	Overhead
Data alone	372	-
Fortran file	388	16
FlatDBMS file	372	0
CDF 22 variables	390	18
CDF 4 variables	569	197
CDF 4 optimized	375	3
CDF 2 optimized	373	1

Thus, the optimized CDF structure has 3.0% overhead on storage as compared with the actual data. By comparison, a Fortran file has 4.3% overhead and a flatfile has 1.9% overhead.

5. Storage Comparisons -- Multi-dimensional Files

The next evaluation of storage requirements used a multi-dimensional file. FlatDBMS is not designed to handle hierarchical files, so for this comparison, only CDFs and Fortran direct access files were used. The data set was comprised of 110 records with each record containing 22 8-byte words and one 4-byte word of various parameters pertaining to that given record plus two 8-byte arrays and one 4-byte array of dimension 29x19 containing the average fluxes (their standard deviations and number of counts, respectively) for each energy level and pitch angle. This data requires 2407 blocks on the VAX.

Since all variables must have the same dimension, the parametric variables also have the dimensions of 29x19 even though they are redundant for the various energies and pitch angles. Without limiting the repetition of these parameters, the CDF (comprised of 26 variables) occupies 24,174 blocks. However, by constraining the 23 parameters, this CDF needs only 2415 blocks. Thus, there are only 8 blocks of overhead for the data itself.

Again no account has been taken of the storage requirements of the header information. This is due to the flexibility of the CDF. The exact storage requirements of the header is dependent upon the needs of the user as was described in the previous section. Note, that for this CDF, the 26 variables require 12 blocks for the .CDF directory file.

For comparison purposes, four Fortran direct access files of various record structures were written. Their particular structures are as follows.

File 1 had 110 records (record length of 2800 longwords, where a longword equals 4 bytes) with each record containing 23 parametric values and three 29x19 data arrays, one for averages, standard deviations, and counts. This took up 2407 blocks on the VAX.

File 2 contained 440 records (record length of 1102 longwords) with the parametric values in record one, the average array in record two, the standard deviation array in record three, and the count array in record four. This four record pattern was then repeated for the 110 times. This structure used 3789 blocks (due to zero filling when the record did not actually contain 1102 longwords).

File 3 was comprised of 6380 records (record length of 58 longwords) with the 23 parametric values in record one, the 29 average values in records two through 20, 29 standard

deviation values in records 21 through 39, and 29 count values in records 40 through 58 (i.e., the 29x19 arrays were broken into 19 records of 29 values each). This 58 record pattern was then repeated for the 110 times. This file occupied 2891 blocks.

File 4 was composed of 184,360 records (record length of 2 longwords) with each record containing one value. Thus, records one through 23 each had one parametric value, records 24 through 574 each had one average value, records 575 through 1125 each had one standard deviation value, and records 1126 through 1676 each had one count value. This 1676 record pattern was then repeated for each of the 110 times. This file needed 2881 blocks.

File Type	Storage Blocks	Overhead
Data alone	2407	-
CDF 26 variables	24174	21767
CDF 26 optimized	2415	8
File 1	2407	0
File 2	3789	1382
File 3	2891	484
File 4	2881	474

Thus, the optimized CDF file required 0.9% more storage than the data alone or the long record Fortran direct access format.

6. Preliminary Timing Comparisons

Bear in mind for this discussion of system requirements, particularly the CPU time requirements, that this investigation was done on a VAX 8650 with a Floating Point Unit (FPU) and 48 Mbytes of main memory. The operating system used was VAX/VMS version 4.7. As described in the CDF Implementer's Guide, the process quotas were set such that the paging file quota was 69063 (the minimum quoted in the Guide was 20000), the enqueue quota was 400, and the open file quota was 120.

The first set of timing runs were done on the previously discussed multi-dimensional files. The optimized CDF performance was compared to that of the four Fortran direct access files. For this test, a set of random data values were selected. The CPU time necessary to read these values and print them out to the screen was obtained using VAX/VMS library calls to the timer. The longer records required fewer read statements. Thus, each file required a different number of reads to get at the same data. Along with the CPU time, timer returned elapsed time, buffered I/O, direct I/O, and page faults.

A series of test runs were conducted accessing the same values in 45 different ways with each test run three times. These three tries were then averaged. The buffered I/O and direct I/O did not vary with each of the 45 runs, but the elapsed time, CPU time, and faults varied somewhat. The 45 different accessing patterns did not show any particular trends in enhanced or degraded timing. This is to be expected in a direct access file structure as opposed to a sequential structure. Thus, the following comparisons will be with respect to all cases for a given file format and not to any particular subset of cases.

The buffered I/O was the same for all of the Fortran files, but was slightly higher for the CDF file. The direct I/O was nearly the same for Files 1 through 3, but was an order of magnitude higher for File 4. The CDF file was only 3-4 times higher than the first three files.

The page faults for Files 3 and 4 were very similar with the number in the mid-thirties for the most part. File 2 was slightly higher, in the low forties. File 1 was a bit higher still with the number of page faults into the mid-fifties per trial. However, the CDF file was significantly higher with values in the low to mid-1100s.

Required CPU time for Files 1 through 3 were very similar with File 1 requiring an average of .36 seconds, File 2 requiring .35 seconds, and File 3 requiring .40 seconds. File 4 requirements were much higher than these, with an average of 1.68 seconds. The CDF also required more than the first three, but not as much as four with an average of 1.33 seconds. The number of data points accessed for this comparison was relatively small. There is a question whether or not the CPU requirements here are representative of a full-scale processing program. Thus, another test was conducted using a complete data set.

	CPU(min)	BUFIO	DIRIO	FAULTS
CDF:	1.33	266	47	1137
File 1:	0.36	240	10	55
File 2:	0.35	240	15	43
File 3:	0.40	240	15	36
File 4:	1.68	240	135	34

7. Full-Scale Timing Comparison

For this comparison, it was decided that a comparison to FlatDBMS would be useful since it too is a generalized data format package. Thus, a flatfile rather than an hierarchical file was selected as the test database. The same file used for the previous storage requirements was used here as well. The Bx, By, and Bz data taken at the seven stations was taken at five second intervals over an interval of approximately three hours. Using a smoothing routine, SMOOFT, from Numerical Recipes [Press et al., 1986] the five-second data was smoothed into one-minute averages.

The FlatDBMS file was written into a CDF structure with four variables and optimized to eliminate redundancy (these two files are discussed in a previous section: Storage comparisons -- Flatfile). Then this data set was written into two different Fortran direct access files with the following record structures.

The Fortran direct access file with long records (FDA(long)) has 2068 records (1 record/time sample) each containing,

Time (8 bytes),
Bx -- for 7 stations (4 bytes),
By -- for 7 stations (4 bytes),
Bz -- for 7 stations (4 bytes).

Thus, the requisite storage is 372 blocks.

The Fortran direct access file with short records (FDA(short)) has (2068 time samples)(22 recs/time sample) = 45,496 records with the following structure,

Record 1: Time (8 bytes),
Record 2 - 8: Bx1 - Bx7 (4 bytes),
Record 9 - 15: By1 - By7 (4 bytes),
Record 16 - 22: Bz1 - Bz7 (4 bytes).

Thus, the required disk space is 711 blocks, since each record in the file must have the same fixed length and the longest record is 8 bytes.

Each file was read in with the data placed in various arrays. Each array was then smoothed and the subsequent arrays were written out to a file of the same type as was read. The results are as follows:

	CPU(sec)	BUFIO	DIRIO	FAULTS
FF:	17.66	37	79	764
	17.32	37	79	758
	17.16	37	79	760
CDF:	43.69	52	246	1041
	43.70	52	242	892
	43.69	52	243	884
FDA(long):	18.72	35	75	726
	18.63	35	75	718
	18.56	35	75	730
FDA(short):	34.24	45	1511	721
	34.11	45	1512	718
	33.52	45	1511	725

Clearly, the CDF requires much more time than the other formats, about 2.5 times longer than a flatfile or a Fortran direct access file with long records. In fact, the one which is closest in the time requirements is the direct access which has one piece of data per record. The CDF is also based on this one variable per record format, which is why the time required is so high. Each item requires a call to the CDF_var_get subroutine, whereas the long record direct access file gets 22 variables per read.

8. Comments

A generalized data format has many virtues: easier data transfer between various users, reduced application development time, and quicker turn-around time to obtain results. All of which are designed to simplify and speed data analysis. Of the two packages discussed here, CDF is clearly more flexible and is adaptable to a broader range of data sets, even though the FlatDBMS processes data faster. If CPU time is truly of the essence, one should probably rely on the highly specific Fortran direct access file structure; however, gains in CPU time may be offset by application development time when using a variety of data sets.

To improve the processing time of the CDF structure, there should be a way to bring arrays of a variable back from the database with one subroutine call rather than having to make a call for every single piece of information. This alone could bring the CDF processing time within reasonable proximity to that of a Fortran direct access file. Note that this evaluation has been based on the VMS Fortran version 1.0 of the CDF routines. We understand (private communication, Lloyd Treinish) that the next version of the CDF from NSSDC will be able to access more than a single element at a time. This next version will be in C, but Fortran bindings will be available so that the routines may still be used within Fortran source code.

For data sets which are anticipated to have a wide range of distribution, the increased processing time may well be entirely offset by the simplification of distribution for a standardized data set and by a standard set of mature routines with which to analyze the data. The time saved in application software development could well be significant. Both the FlatDBMS and CDF packages are easy to learn and use. One can become reasonably proficient in using either one in a day or two. The CDF system is a little bit easier to handle, but there is not a large difference in the learning curve between the two.

As mentioned previously, there is a drawback to being denied access to the source code of these routines. For example, in writing a routine to translate a FlatDBMS file into a CDF file, there was a problem with the CDF not checking to see which unit numbers were already in use for the FlatDBMS files. The FlatDBMS system checks for this, however, it limits the number of files open at any given time to six. The CDF already had opened more than twenty files. Thus, either the CDF needed to be altered to check for previously opened units or the FlatDBMS needed to be altered to allow more files to be opened simultaneously. Fortunately, the FlatDBMS source code was available so this problem could be solved.

Again, we have learned (private communication, Lloyd Treinish), that this problem is resolved by the upcoming C version of the code. Nonetheless, it is this sort of problem which necessitates the availability of the CDF source code. NSSDC is aware of this problem and its resolution is under consideration. Software licenses may become available in the future. Additionally, concerns have been raised about the need to know more about the physical structure of the files in memory. This information is not provided in the Implementer's Guide, but it is available upon request from NSSDC.

We have attempted to provide an unbiased evaluation of CDFs by examining CDF features which are relatively easy to quantify such as storage requirements and access speed. However, it is important to keep in mind that CDFs have a number of other virtues which are more difficult to quantify, but which we have only briefly mentioned in this document. The self-documenting data structure, the capability for handling diverse multi-dimensional data, and the advantages for facilitating data exchange and software application development should not be ignored when making a decision about adopting the CDF data structures.

9. Acknowledgements

We are grateful to Lloyd Treinish at NSSDC for providing the CDF code, documentation, and examples, and for answering our questions on numerous occasions.

References

Gough, M.L., Goucher, G.W., and Treinish, L.A., NSSDC CDF Implementer's Guide (DEC VAX/VMS), Version 1.1, National Space Science Data Center, Greenbelt, MD, August 1988.

Press, W.H., Flannery, B.P., Teukolsky, S.A., and Vetterling, W.T., Numerical Recipes, The Art of Scientific Computing, Cambridge University Press, Cambridge, England, 1986.

Smith, A.Q. and Clauer, C.R., "FlatDBMS: A Versatile Source-Independent System for Digital Data Management", EOS, 67, n15, pp 188-189, April 15, 1986a.

Smith, A.Q. and Clauer, C.R., "FlatDBMS: A Flexible Source-Independent Data Management System for Scientific Data", STAR Lab Report D106-1984-1, Stanford University, Stanford, CA, May 1986b.

Treinish, L.A. and Gough, M.L., "A Software Package for the Data-Independent Management of Multi-Dimensional Data", EOS, 68, n28, pp 633-635, July 14, 1987.