A Software Package for the Data-Independent Management of Multidimensional Data

PAGES 633–635

Lloyd A. Treinish and Michael L. Gough

National Space Science Data Center, NASA/Goddard Space Flight Center, Greenbelt, Md.

Introduction

The National Space Science Data Center (NSSDC) of the National Aeronautics and Space Administration (NASA) Goddard Space Flight Center (GSFC) provides access to data from and information about a plethora of scientific experiments from a variety of disciplines. To help fulfill this mission, NSSDC has developed a software package that supports a self-describing data structure. This structure, called the Common Data Format (CDF), provides true data independence for applications software that has been developed at NSSDC. Scientific software systems at NSSDC use this construct so that they do not need specific knowledge of the data with which they are working. This permits users of such systems to apply the same functions to different sets of data. This data-independent concept was first introduced at NASA in support of atmospheric and climatic research via the Pilot Climate Data System (PCDS), a scientific information system and analysis system developed at NSSDC [*Treinish,* 1984; *Reph et al.,* 1986]. The users of such data-independent NSSDC systems as the PCDS rely on their own knowledge of different sets of data to interpret the results, a critical feature for the multidisciplinary studies inherent in the earth and space sciences. Such CDF-based software can use the information available through the CDF software package to inform a user about contents, history, and structure of the data that are supported in a given CDF.

Background

The CDF was originally conceived, designed, and implemented in 1982 as a means of transferring data in many different formats to a uniform format for use in generic data display software via computer graphics in the PCDS [*Treinish,* 1984]. This initial implementation represented a prototype for a data-independent storage structure, in which only some of the CDF concepts described in the following sections were made available.

The success of the CDF in providing a mechanism for the uniform treatment of a wide assortment of disparate climate data sets resulted in its being hailed as one of the outstanding features of the PCDS. This common format for the storage and transfer of climate data made it easy to develop generic graphical and analytical tools that were based upon the CDF but were readily applicable to a collection of data accumulated from a broad spectrum of climate-related experiments. However, as the functional capabilities of the PCDS expanded in scope after its initial implementation, the original prototype CDF design was pushed beyond its limits in order to accommodate greater data processing requirements. Hence the need arose for the generalization of the CDF concept (outlined herein) to meet the expanding PCDS requirements. The implementation of the generalized CDF was completed in 1986.

The success of the CDF approach in meeting the data management requirements of the PCDS came to the attention of committees that were responsible for planning other NSSDC systems that needed to solve similar problems (i.e., the uniform treatment of data collected from a diverse range of sources). Chief among these systems is the Pilot Land Data System (PLDS), which has data management requirements analogous to the PCDS but must also handle large volumes of satellite image data [*Campbell et al.,* 1986]. Furthermore, the CDF is now the standard vehicle for the collections of data that are used in the Coordinated Data Analysis Workshops (CDAW) in support of solar-terrestrial physics applications [*Vette et al.,* 1982; *Manka,* 1986]. CDAW has some data management requirements similar to the PCDS but is more oriented toward detailed data analysis. The use of such a common format has made it possible to share data between systems, to combine diverse data sets, and to transport software modules from one system to another.

Cover. In 1979, Columbia Glacier (in Alaska) was terminating in nearly the same position as it had since at least 1899, when it was mapped by G. K. Gilbert. The embayments in the 1979 terminus are precursors to a drastic retreat that began in the early 1980s. By 1986 the glacier had receded about 2 km from the terminal moraine that is clearly indicated by the sharp discontinuity in iceberg density. Water depth over the terminal moraine is no more than 22 m, water depth near the 1986 terminus is about 300 m, and water depth in Columbia Bay (toward the viewer in this photo) is about 200 m. Retreat of about 30 km is expected over the next several decades.

The retreat of Columbia Glacier was just one of the many glacier-related topics discussed at the May 1986 AGU Chapman Conference on Fast Glacier Flow. For a report of that meeting, see page 638 (Photograph 79L3–028, August 22, 1979, taken by Austin Post, U.S. Geological Survey, Tacoma, Wash.)

Description

The CDF, through its software package, provides to the applications programer a mechanism for uniformly viewing data of interest via a data structure that is oriented to the user of the data (i.e., a scientist). It is a conceptually simple framework for the creation of generic applications (e.g., graphical displays, statistical analysis) and transparent (i.e., usable without the user being aware of the intervening mechanisms), discipline-oriented or user-chosen views of data. It is a uniform structure for the distribution of self-descriptive data, which can be supported by analysis software. This mechanism for the flexible organization of interdisciplinary data into generic multidimensional structures consistent with potential scientific interpretation provides a simple abstract conceptual environment for the scientific applications programer who works with data, but it also encourages the decoupling of data analysis considerations from those of data storage. The developer of CDF-based applications can easily create software that permits a user to slice data across multidimensional subspaces. However, the CDF is not a standard format that allows programers to "grovel" in the bits. Neither is it a mechanism for programers to write messy Fortran formats, and it is not a structure for storing and translating obscurely packed data formats between strange operating systems. Finally, it is not a format with which programers have to consider low-level input/output tasks.

The hallmark of the Common Data Format concept is data set independence. This independence is achieved by means of an internal format, containing its own data dictionary, which is, in effect, a data base system. In other words, a CDF defines its own format. This self-defining property makes it possible for the CDF to be used for data from a wide variety of disciplines.

A CDF is therefore composed of two classes of information: the scientific data themselves and the information defining that data and describing its organization within the CDF structure. The descriptive information (or metadata), as well as the data themselves, can be accessed by means of standard software routines. These CDF interface routines give programers an abstract view of the contents of a CDF while relieving them from the burden of physically packing data into files or translating the metadata to ascertain file contents. As such, these routines are analogous to the access routines provided by a typical data base management package.

The concept of using a data dictionary to describe the contents of a data file is not new for the purpose of achieving a data-independent transportable standard, especially in the geophysics community [*Thomas and Guertin,* 1981]. However, the CDF differs from those earlier formats by being oriented toward the researcher's (rather than the programer's) view of the data. The CDF interface routines not only relieve the scientist/programer of low-level burdensome tasks but in fact establish a concept of data organization consistent with the scientific interpretation of the data.

0096–3941/87/6828–633$1.00

The most important difference between the CDF and conventional data format standards, such as the FLATDBMS [*Smith and Clauer,* 1984, 1986] and its predecessor, Block Data Set [*McPherron,* 1976], is in the nature of the data descriptions maintained within the CDF and of its supporting software (see the implementation section below). It should be noted that although there are similarities between CDF and earlier efforts, such as FLATDBMS, CDF was developed independently of them. Each of these data descriptions in the CDF not only defines the name of each data variable and its units (e.g., TEMPERATURE [DEGREES KELVIN]), but also specifies the organization of the individual values of the variable into a construct consistent with the interpreted dimensionality of the data ensemble. Although FLATDBMS, for example, does maintain some internal data descriptions similar to CDF, such metadata does not include the definition of multidimensional (i.e., nonscalar) constructs. CDF provides the ability to define such multidimensional structures, which are a mechanism for viewing a data ensemble that constitutes some conceptual entity of interest to a user (e.g., an atmospheric temperature profile, that is, a collection of temperatures at various levels in the atmosphere). Block Data Set, for example, supports multidimensional structures, but unlike CDF, it is limited only to the sequential access of multiple variables, which are assumed to be sampled at equidistant intervals [*McPherron,* 1976].

The simplest such data construct would represent one dimension of data, a collection or vector of numbers. The next level would imply two dimensions of data as two parallel vectors, such as an atmospheric profile with a vector of values and a vector of levels, each of which corresponds to a value. A three-dimensional construct implies a matrix of values and a matrix of auxiliary data, such as a map of values at specific latitude-longitude locations. Another example would be the combination of two two-dimensional constructs: a time history of values and a profile of values to yield a profile history; for example, a collection of information as a function of time and atmospheric height.

Table 1 shows this progression of data constructs, provides some climatological examples, and illustrates ways of viewing such entities graphically. It should be emphasized that although the examples are primarily from the atmospheric sciences, the techniques apply to regularly structured data from any discipline.

TABLE 1. Multidimensional Data Constructs

Dimensions of Data Supported	Data Type	Graphic Examples
1	*flat* data	histogram
2	time histories, atmospheric profiles, zonal means, particle spectra	*X–Y* plot
3	grids/images, zonal profiles, zonal histories, profile histories, spectra histories	contour plot, 3-D surface, color image, *X-Y-Z* plot
4	grid/image histories, gridded profiles, zonal profile histories	animated contours, animated 3-D surface, 3-D surface with color
5	gridded profile histories	animated 3-D surface with color
.	.	.
.	.	.
.	.	.

Conceptual Organization

As Table 1 suggests, the number of interesting multidimensional data constructs is quite large, even when the field of interest is restricted to climatology. Moreover, the CDF must be capable of storing data ensembles from a number of other disciplines, including (at least) earth science, solar-terrestrial physics, oceanography, planetary astronomy, and astrophysics. Clearly, it would not be practical to design such data constructs into the CDF individually on a one-by-one basis. Instead, the CDF incorporates a generic data handling mechanism that applies universally to a class of multidimensional data constructs.

The goal of handling such a diverse collection of data objects creates the potential for an individual CDF to become an assortment of entities of various dimensionalities and sizes. The correlation of objects of different dimensionalities within a single CDF could be ill-defined, relying on higher-level data-dependent software to resolve potential ambiguities. This problem is solved by specifying that a CDF be built as a multitude of similar structures. Each CDF is (conceptually) composed of repetitions of a single *n*-dimensional grid structure, where the number of dimensions and size of each dimension in such a structure is arbitrary but is defined by the programer at the time of initial CDF creation. The overall CDF data ensemble is generated by propagation of this grid structure from variable to variable and from record to record, with each occurrence of the grid carrying its own collection of data values. Data values are correlated between different occurrences of the grid by means of grid indices: for example, a data value in one occurrence of the grid is correlated to that specific data value in another occurrence of the grid that has identical indices.

The dimensionality of this basic grid structure, the number of variables, and the number of records can all be specified independently. In other words, the CDF is constructed from fundamental building blocks, whose size imposes no restriction on the number used or on their arrangement. As a consequence, there is no a priori correlation between the dimensionality of the building block (basic grid structure) and the dimensionality of the data ensemble as a whole. To avoid confusion, the dimensionality of the basic grid structure will be referred to as the CDF rank. The CDF rank then is the dimensionality of its basic building block or the number of dimensions in the basic grid structure.

Figure 1 shows the concept of CDF data organization, a uniform multidimensional block structure for a CDF of rank two. The simplest or degenerate case of a CDF basic grid structure is one of rank zero; it contains only a single data value, or scalar. This can be envisioned by substituting a single data value for each two-dimensional grid in Figure 1. Such a CDF is virtually identical to the FLATDBMS structure [*Smith and Clauer,* 1984]. It should be noted that although the dimensionality of each variable in a CDF may not be the same, a basic grid structure is constructed to encompass all of the variables, and its rank is assigned accordingly. Despite the sequential layout of the data in Figure 1, each element of each grid and of each record can be accessed in any order.

The CDF conceptually supports an organizational hierarchy that, at the basic level, classifies units of data into elements or variables, each of which corresponds to a single observable parameter. These variables can be described by attributes. A single atom of one of these variables, or a single observed value or datum, can be visualized at the grid points of the *n*-dimensional basic grid that is invariant within a CDF. In Figure 1, each basic grid block contains 25 atoms, and hence there are 25 values for each variable. Basic grids for a group of variables are collected into a record (i.e., one block for each variable). A collection of records constitute a data ensemble. A variable is referenced by its mnemonic, which points to the corresponding metadata (e.g., attributes) in a CDF.

The CDF is composed of more than just a data ensemble. There is a data dictionary and attribute table that contain the various aforementioned characteristics and ancillary information that define the data ensemble completely. The data dictionary specifies whether or not each variable varies with respect to records (record variance) or to the individual dimensions of the basic grid structure (dimensional variance). The attribute table sup-

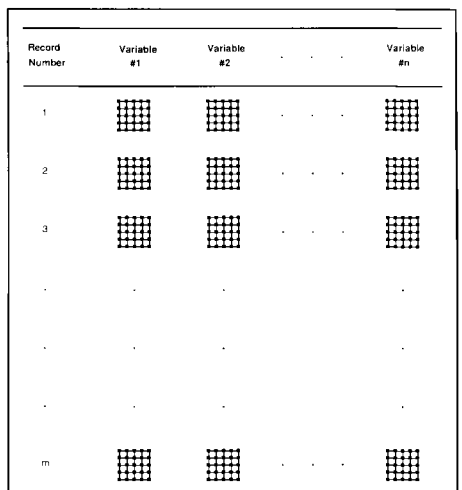


Fig. 1. Conceptual view of a CDF data ensemble with rank = 2.

ports specific information about variables, such as name, mnemonic, scientific units, type (e.g., real*4, integer*2 in the Fortran sense), range, resolution, and display format, as well as global information about the entire ensemble. This global information might include (for example) statistics, which could include minimums and maximums, and text, which can be used to support documentation. To help illustrate these concepts, a simple example is presented in the following section.

An Example CDF Structure

Table 2 contains a simple data ensemble that can be used to illustrate the various aforementioned CDF concepts. This ensemble contains a collection of temperature measurements at different times and locations. The accompanying box (Description of One Variable (Attributes)) shows an example of the type of general descriptions or attributes that CDF supports in its data dictionary for data elements or variables. If one examines the data ensemble in detail, it becomes apparent that it contains more than a simple flat structure, despite its given organization. For example, time is organized into blocks of four identical values. Latitude and longitude are each cyclic, with two fixed values (+30, +40) and (−165, −150), respectively. Hence the temperature values are organized into a 2 × 2 grid for each observation, where longitude and latitude represent the dimensions of that grid. CDF supports this type of data structure implicitly. The 2 × 2 temperature grid actually implies a uniform 2 × 2 virtual block structure (i.e., a CDF of rank 2) for the entire CDF. In addition, the CDF software can take advantage of information about such data structures to conserve storage space. This internal elimination of redundancy is illustrated in Figure 2 and the accompanying Table 3. The CDF specifications presented in Table 3 show the information that the programer must provide in order to eliminate such redundant data storage. In this example, latitude and longitude are invariant with respect to record number and hence are stored only once. The uniform block structure implies that the values of the elements that are invariant with record number (i.e., latitude and longitude) appear to be duplicated for successive records and that the values of the elements that are invariant with respect to a basic grid dimension (i.e., time) appear to be duplicated across that dimension. CDF portrays to the programer a uniform block structure in which equal random access to all elements is provided, while any redundant storage inherent in that structure is eliminated for its physical storage. Although this simple example shows a time series of data organized into a single grid structure, CDF can just as easily handle non–time series data organized into complex grids.

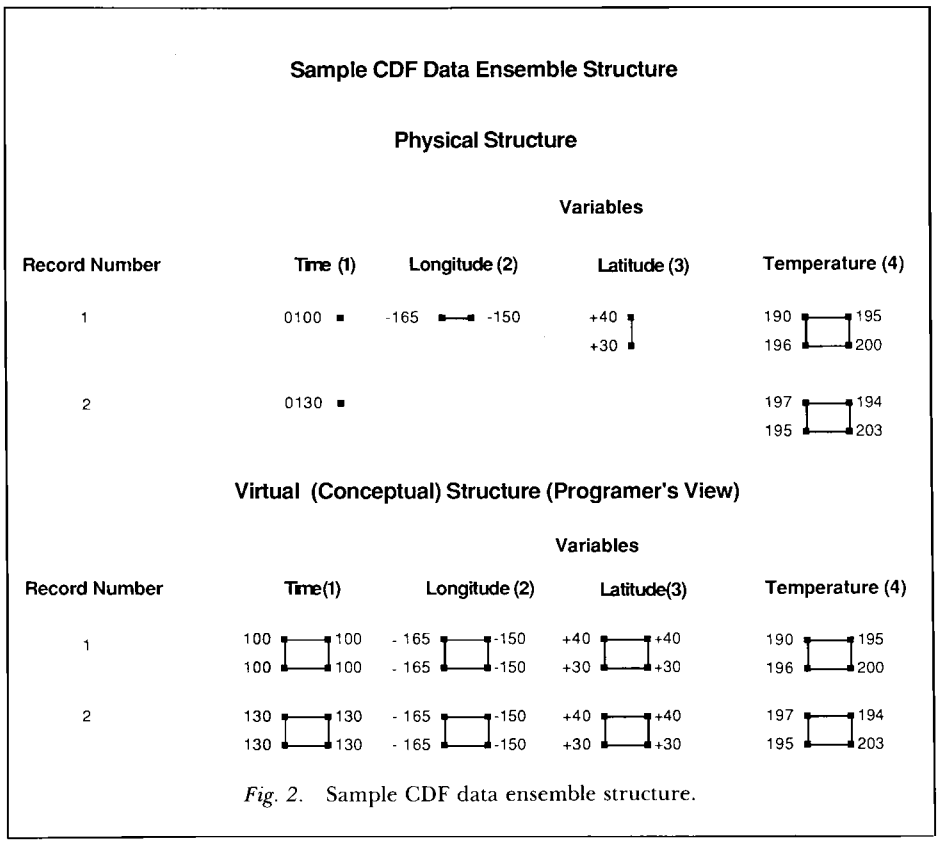


Fig. 2. Sample CDF data ensemble structure.

Implementation

The CDF isolates the details of the structure of a data set from a user of such data in any applications software. Therefore the programer of such applications only needs to know about the collection of CDF operations. These operations, which are maintained in the CDF interface routine library, permit a programer to create, access, fill, extract, and query the data and variable attributes in a CDF. The programer does not need to know the details of the CDF storage nor the underlying software structure because the CDF is implemented as a data abstraction [*Shaw*, 1984; *Berzins et al.*, 1986]. This isolation permits enhancements to the CDF implementation as new software and hardware technology permit, without requiring changes to applications software. The user simply perceives improved performance or functionality (in other words, the CDF structures and implementation are transparent to the user). In addition, the CDF concept is extensible in the programer's perspective by the addition of new operations. Hence the interface routine library or the CDF software package is a toolbox of programing primitives for managing multidimensional data ensembles; it provides a simple abstract view for random access of arbitrary blocks of data. Any analysis or other applications capabilities must be built into higher-level software that employs CDF. The programer that utilizes the CDF data abstraction views the CDF interface routine library as consisting of 13 operations that address the basic features of the CDF: dictionary, structure, data ensemble, summary statistics, and documentation, as well as general file management. The library represents the Fortran language bindings for these operations as implemented for Digital Equipment Corporation (DEC) VAX/VMS computer systems. These abstract routines are designed to make it easy for a programer to utilize data in terms of CDF, independently of the complexity of the data. Optimization for high-performance (minimal use of memory, central processing unit resources, and input/output operations) in the VAX/VMS environment has been incorporated within the CDF software to eliminate the overhead that is typically present in data management systems that use simple sequential files (e.g., Block Data Set), but it is also isolated in a way which simplifies future porting to other operating systems. For example, the physical structure of a CDF on VAX/VMS systems consists of $n + 2$ binary random access files, where n is the number of variables in the CDF. The other two files contain the data dictionary with its related statistics and documentation (i.e., metadata) and the definition of the data structure, respectively. However, these files are transparent to the user and appear integrated as a single CDF via the CDF software. In addition, the CDF software employs a high-speed caching algorithm similar to those that

TABLE 2. Example CDF Structure: Data Ensemble

	Variables			
Record Number	Time (1)	Longitude (2)	Latitude (3)	Temperature (4)
1	0100	−150.	+30.	200.
2	0100	−150.	+40.	195.
3	0100	−165.	+30.	196.
4	0100	−165.	+40.	190.
5	0130	−150.	+30.	203.
6	0130	−150.	+40.	194.
7	0130	−165.	+30.	195.
8	0130	−165.	+40.	197.

TABLE 3. Sample CDF Data Ensemble Specifications

Attribute	Variables			
	Time (1)	Longitude (2)	Latitude (3)	Temperature (4)
First Dimension Variance (→)	False	True	False	True
Second Dimension Variance (↑)	False	False	True	True
Record Variance	True	False	False	True
Data Type	Integer * 4	Real * 4	Real * 4	Real * 4

Description of One Variable (Attributes)

Variable mnemonic	TEMP
Variable name	temperature
Variable units	degrees Kelvin
Resolution	0.6
Display format	F7.3
Valid range	170. to 290.

typical virtual memory operating systems (e.g., VAX/VMS) utilize to shuttle data quickly in and out of memory. This ensures that blocks of data that are randomly requested by a programer in one or more CDFs are rapidly available on an as-needed basis.

The operations of the library include routines to create, open, close, delete, and inquire about a CDF; to create and inquire about CDF variables; to enter or extract data from a variable; to create and inquire about CDF attributes; and to enter or extract information from an attribute. It should be noted that once all of the variables have been specified through a "create CDF variable" routine, the programer does not need to keep track of dimension and record variances: The CDF package will manage this information. Although a programer is free to create applications-specific attributes, there are conventions for various CDF attributes that are employed in CDF-based applications at NSSDC (e.g., variable name for labeling a plot axis [*Gough*, 1987]).

Status and Applications

As stated earlier, a subset of the CDF was first implemented in 1982 as a means of providing data-independent access, display, and manipulation of several types of multidimensional data within the prototype version of the PCDS [*Treinish*, 1984]. This implementation provided only some of the features of the complete CDF as a proof of concept. Various software packages were built as CDF applications to permit the PCDS users to easily manipulate and display (through computer graphics) data of interest. The full implementation of the CDF is now complete and has been undergoing alpha testing at NSSDC. The CDF software is also undergoing beta testing at several other sites that support data from a number of different disciplines. (Alpha testing of software implies rigorous utilization in new applications within the software developers' organization, while beta testing involves evaluation by volunteers outside of the original organization.) Within NSSDC, many applications are being built upon this structure, including redesigned operational data access, data manipulation and graphics capabilities within the PCDS, support of data analysis, management and graphics for CDAW, and graphics capabilities for the PLDS. To support these various analysis and display applications, a generic layer called the Virtual Data Table (VDT) has been developed on top of CDF. The VDT provides a "spreadsheet"-type window on any arbitrary two-dimensional subset of a multidimensional structure within a CDF [*Gough*, 1986].

Once the CDF software and generic CDF applications were established at NSSDC, these tools were then used to support a number of scientific research activities in many different disciplines. Dozens of different data sets, in a variety of mutually incompatible formats, have been very easily and quickly converted to CDFs via programs that use the CDF software. Although new data sets are being converted every day through such custom programs, other generic software is being developed to convert one or more classes of different formats or data base representations of data (which may support many different data sets) to CDF. Once any data, whether they are simple or complex in nature, are available in CDF, powerful data-independent applications available at NSSDC in systems such as the PCDS can be used to work with such data for data display (e.g., *x-y* plots, contour plots, histograms, maps, etc.) or for data analysis (e.g., through the Interactive Data Language, or IDL) in a generic fashion [*Research Systems Inc.*, 1986]. Before the advent of CDF, customized applications typically had to be developed to work with complex data in their original format. (*Smith and Clauer* [1984] and *McPherron* [1976] outline notable exceptions to this common situation.) Now, robust CDF-based applications, already available at NSSDC, can be utilized with such data, once they are available in CDF.

Conclusion

The CDF is an abstraction for the data-independent storage and management of multidimensional data, in which the data ensemble appears to be built from multiple occurrences of a single *n*-dimensional block that is consistent with the scientific interpretation of the data (i.e., it provides a user's view of the data rather than that of the programer). The values for different variables are correlated simply by specifying identical record numbers and basic grid indices. Redundant physical storage of data for cyclic variables is eliminated by the specification of record and grid dimension variances. The CDF structure provides flexibility in application and simplicity in use. For example, CDF can support data ranging from simple collections of scalar measurements to very large multispectral images (e.g., from LANDSAT) to complex multidimensional structures. Hence this flexibility and simplicity together yield power for the development of comprehensive, generic systems to support data management and correlative data analysis. This power is needed by the National Space Science Data Center to help fulfill its goal of providing the research community with ready access to easy-to-use, well-documented data. If readers are interested in learning more about CDF, please contact the authors at NSSDC (or NSSDCA::Treinish and NSSDCA::MGough on the Space Physics Analysis Network (SPAN)). Additional documentation about CDF and copies of the CDF Implementer's Guide [*Gough*, 1987], as well as the VAX/VMS implementation of the CDF Software Package for beta testing, are available through the NSSDC Request Office, Code 630.2, NASA/Goddard Space Flight Center, Greenbelt, MD 20771 (or NSSDC::REQUEST on SPAN).

Acknowledgments

The research reported in this paper has been supported mainly by the Office of Space Science and Applications (OSSA) of the National Aeronautics and Space Administration. The authors wish to acknowledge several people from the staff of NSSDC: Clayton E. Wilson for contributions to development of CDF; Mary G. Reph for the management of the software development; and Paul H. Smith for the initiation and the management of the PCDS, under which the early CDF work was done. In addition, the authors wish to thank James L. Green, director of NSSDC, for his continued support of CDF and for the concept of discipline-independent software systems to support scientific research.

References

Berzins, V., M. Gray, and D. Naumann, Abstraction-based software development, *Commun. ACM, 29*, 402, 1986.

Campbell, W. J., P. H. Smith, R. H. Price and L. H. Roelofs, Advancements in land science data management: Pilot Land Data System, *Sci. Total Environ., 56*, 31, 1986.

Gough, M., A generic tool for the generation and display of animated sequences, in *Proceedings of the Third Annual Conference of the TEMPLATE User Network*, Megatek Corporation, San Diego, Calif., 1986.

Gough, M., NSSDC Common Data Format Implementer's Guide, *Rep. NSSDC/SAR 8701*, National Space Science Data Center, NASA Goddard Space Flight Center, Greenbelt, Md., 1987.

Manka, R. H., CDAW Space Data Analysis Progresses, *Eos Trans. AGU, 67*, 1401, 1986.

McPherron, R. L., A self-documenting source-independent data format for computer processing of tensor time series, *Phys. Earth Planet. Inter., 12*, 103, 1976.

Reph, M. G., L. A. Treinish, C. E. Noll, T. D. Hunt, and S.-W. Chen, Pilot Climate Data System Users' Guide, *NASA/GSFC Tech. Memo. 86084* (revised), January 1986.

Research Systems, Inc., *The Interactive Data Language (IDL) User's Guide*, Boulder, Colo., 1986.

Smith, A. Q., and C. R. Clauer, FLATDBMS: A flexible, source-independent data management system for scientific data, *STAR Lab Rep. D106-1984-1*, Stanford Univ., Stanford, Calif., 1984.

Smith, A. Q., and C. R. Clauer, A versatile source-independent system for digital data management, *Eos Trans. AGU, 67*, 188, 1986.

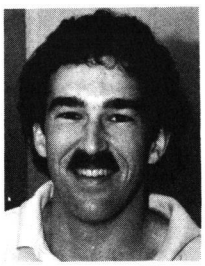
Shaw, M., Abstraction techniques in modern programming languages, *IEEE Software, 1*, 10, 1984.

Thomas, V., and F. E. Guertin, Standardization of computer compatible tape formats

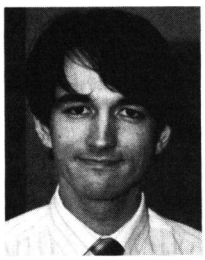
for remote sensing data, paper presented at the 1981 IEEE International Geoscience and Remote Sensing Symposium (IGARSS'81), *IEEE Digest, 2,* 1656, June 1981.

Treinish, L. A., A general scientific information system to support the study of climate-related data, *NASA/GSFC Tech. Memo. 86152,* September 1984.

Vette, J. I., D. M. Sawyer, M. J. Teague, and D. J. Hei, The origin and evolution of the coordinated data analysis workshop process, *IMS Source Book, 112,* 235, 1982.



Lloyd A. Treinish *is a computer scientist at the National Space Science Data Center of NASA Goddard Space Flight Center. He works on the development of advanced data systems for support of scientific applications, as well as studying space and atmospheric phenomena. His research interests range from computer graphics, data storage structures, data representation methodologies, computer user interfaces, and data analysis algorithms to middle atmosphere electrodynamics, planetary astronomy, and climatology. A 1978 graduate of the Massachusetts Institute of Technology, with a S.M. and a S.B. in physics and a S.B. in earth and planetary sciences, Treinish has been at NASA since 1979. He is a member of AGU, the Association for Computing Machinery (ACM), ACM SIGGRAPH, the Planetary Society, and the IEEE Computer Society.*



Michael L. Gough *is a senior software engineer with Science Applications Research Corporation at the National Space Science Data Center. He works on the development of advanced data systems for support of scientific applications. He has designed both generic software tools that produce animated graphics of scientific data and advanced data structures for scientific data. He has developed virtual memory and network communications software for NASA's Massively Parallel Processor, as well as real-time satellite tracking and data ingest software for weather forecasting systems in the People's Republic of China and Argentina. Gough has been with Science Applications Research Corporation since 1983.*