US Army Corps of Engineers
Hydrologic Engineering Center

Generalized Computer Program

HECLIB

Volume 2: HECDSS Subroutines

Programmer's Manual

May 1991

CPD-57

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** May 1991	**3. REPORT TYPE AND DATES COVERED** Computer Program Document No. 57

4. TITLE AND SUBTITLE HECLIB Volume 2: HECDSS Subroutines Programmer's Manual	**5. FUNDING NUMBERS**
6. AUTHOR(S) CEWRC-IWR-HEC	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) US Army Corps of Engineers Institute for Water Resources Hydrologic Engineering Center 609 Second Street Davis, CA 95616-4687	**8. PERFORMING ORGANIZATION REPORT NUMBER** CPD-57
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER** N/A
11. SUPPLEMENTARY NOTES N/A	
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for Public Release. Distribution of this document is unlimited.	**12b. DISTRIBUTION CODE**

13. ABSTRACT *(Maximum 200 words)*

Programmer's manual for computer programmers to interface FORTRAN application programs to HEC's Data Storage System (HECDSS or DSS). The DSS is a database system that was developed to meet needs for data storage and retrieval for water resource studies. The system enables efficient storage and retrieval of time series (such as precipitation hyetographs or hydrographs of stage, discharge, etc.) and other data-types for which storage in blocks of contiguous data elements is most appropriate. The DSS consists of a library of FORTRAN subroutines, which can be readily used with application programs to enable retrieval and storage of information. The current application programs include the widely used program Flood Hydrograph Package (HEC-1) and the Expected Annual Damage (EAD) program. In addition, approximately 17 DSS utility programs have been developed. A number of these programs are for data entry, for example from the USGS WATSTORE database, or from NWS precipitation data files. Other utility programs include a powerful graphics program, a report generator, and a program for performing mathematical transformations.

14. SUBJECT TERMS HECDSS, DSS, Time-Series Data, Database	**15. NUMBER OF PAGES** 296
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	**18. SECURITY CLASSIFICATION OF THIS PAGE** UNCLASSIFIED	**19. SECURITY CLASSIFICATION OF ABSTRACT** UNCLASSIFIED	**20. LIMITATION OF ABSTRACT** UNLIMITED

HECLIB
Volume 2: HECDSS Subroutines

Programmer's Manual

May 1991

US Army Corps of Engineers
Institute for Water Resources
Hydrologic Engineering Center
609 Second Street
Davis, CA 95616

(530) 756-1104
(530) 756-8250 FAX
www.hec.usace.army.mil

CPD-57

HECLIB
Volume 2: HECDSS Subroutines
Software Distribution and Availability Statement

Table of Contents

Chapters

1 **Introduction**
- 1.1 Background 1-1
- 1.2 DSS Contrasted with Other Database Systems 1-2
- 1.3 General Information 1-2
- 1.4 A Short Description of How DSS Works 1-4
- 1.5 Programming with DSS 1-5
  - 1.5.1 Opening and Closing DSS files 1-5
  - 1.5.2 The IFLTAB Array 1-6
  - 1.5.3 DSS Subroutines 1-6
  - 1.5.4 Message Control 1-7
- 1.6 Typical Order of Calling DSS Subroutines 1-7
- 1.7 Machine Specifics 1-8
  - 1.7.1 HARRIS Computers 1-8
  - 1.7.2 Microcomputers Using Microsoft® FORTRAN 1-8
  - 1.7.3 Microcomputers Using Lahey® FORTRAN 1-9
  - 1.7.4 UNIX Operating Systems 1-10

2 **General Subroutines**
- 2.1 ZOPEN – Open a DSS File 2-2
- 2.2 ZCLOSE – Close a DSS File 2-5
- 2.3 ZFNAME – Add Filename Extension and Determine if the File Exists 2-6
- 2.4 ZDTYPE – Determine a Record's Data Type and if it Exists 2-8
- 2.5 ZSET – Set DSS Parameters 2-11
- 2.6 ZINQIR – Inquire About DSS Parameters 2-17
- 2.7 ZFVER – Get a DSS File's Version 2-20

3 **Pathname Manipulation Subroutines**
- 3.1 ZPATH – Construct a Pathname 3-2
- 3.2 ZUPATH – Determine a Pathname's Part 3-4
- 3.3 ZUFPN – Spilt a Pathname into Separate Parts 3-6
- 3.4 ZGPNP – Get Pathname Parts 3-8
- 3.5 ZCHKPN – Check a Pathname 3-12

4 **Time Series Subroutines**
- 4.1 ZRRTS – Retrieve Regular-Interval Time Series Data 4-3
- 4.2 ZRRTSX - Retrieve Regular-Interval Time Series Data (Extended Version) 4-8
- 4.3 ZSRTS – Store Regular-Interval Time Series Data 4-14
- 4.4 ZSRTSX – Store Regular-Interval Time Series Data (Extended Version) 4-17
- 4.5 ZRITS – Retrieve Irregular-Interval Time Series Data 4-22
- 4.6 ZRITSX - Retrieve Irregular-Interval Time Series Data (Extended Version) 4-26
- 4.7 ZSITS – Store Irregular-Interval Time Series Data 4-32

Table of Contents (continued)

Chapters

4 **Time Series Subroutines (continued)**
4.8 ZSITSX - Store Irregular-Interval Time Series Data (Extended Version) 4-37
4.9 ZGINTL – Get Time Series Interval 4-42
4.10 ZOFSET – Determine the Time Offset of Time Series Data 4-44

5 **Paired Data Subroutines**
5.1 ZRPD – Retrieve Paired Data 5-2
5.2 ZSPD – Store Paired Data 5-7

6 **Text Subroutines**
6.1 ZRTEXT – Retrieve Text Data (Into a File) 6-2
6.2 ZRTXTA – Retrieve Text Data (Into an Array) 6-5
6.3 ZSTEXT – Store Text Data (From a File) 6-8
6.4 ZSTXTA – Store Text Data (From an Array) 6-11

7 **Catalog and Tag Subroutines**
7.1 ZOPNCA – Open a Catalog File 7-3
7.2 ZCAT – Catalog a DSS File 7-6
7.3 ZRDCAT – Read Pathnames from a Catalog File 7-10
7.4 ZRDPAT – Read Pathnames from a Catalog File by Reference Number 7-14
7.5 ZTAGPA – Get Pathnames from Tags 7-17
7.6 ZRETAG – Change of Record Tag 7-21
7.7 ZSTAGS – Set the Tag Scheme for a DSS File 7-22
7.8 ZRTALL – Change All Record Tags in a DSS File 7-24
7.9 Example of Obtaining Pathnames from References on a Command Line 7-25
7.10 Example of a Catalog Display Subroutine 7-30

8 **General Read/Write Subroutines**
8.1 ZREAD – Read an Individual Record 8-2
8.2 ZREADX – Read an Individual Record (Extended) 8-5
8.3 ZRDBUF – Read an Individual Record in a Buffered Mode 8-7
8.4 ZWRITE – Write an Individual Record 8-10
8.5 ZWRITX – Write an Individual Record (Extended) 8-13
8.6 ZWRBUF – Write an Individual Record in a Buffered Mode 8-16

9 **Utility Subroutines**
9.1 ZSTFH – Stuff the User Header Array 9-2
9.2 ZUSTFH – Disassemble the User Header Array 9-6
9.3 ZCHECK – Check if a Record Exists 9-10
9.4 ZRECIN – Display Information About a Record 9-12
9.5 ZFILST – Display Information About a DSS File 9-14
9.6 ZCOREC – Copy a Record 9-15
9.7 ZCOFIL – Copy a DSS File 9-18

Table of Contents (continued)

Chapters

9 **Utility Subroutines (continued)**
- 9.8 ZRENAM – Rename a Record 9-20
- 9.9 ZDELET – Delete a Record 9-21
- 9.10 ZUNDEL – Undelete a Record 9-22
- 9.11 ZUDALL – Undelete All Records in a DSS File 9-23
- 9.12 ZDEBUG – Display Coded Information from the File or the IFLTAB Array 9-24

10 **Data Compression Subroutines**
- 10.1 ZSCOMP – Set Data Compression for a Record 10-3
- 10.2 ZDCINF – Get Data Compression information for a Record 10-5
- 10.3 ZSETCI – Set Default Data Compression for a DSS File 10-7
- 10.4 ZPRTCI – Print the Default Data Compression for a DSS File 10-10

11 **Outdated Subroutines**
- 11.1 ZFPN – Form DSS Pathname 11-2
- 11.2 ZGTDTS – Get Regular-Interval Time Series Data 11-4
- 11.3 ZPTDTS – Put Regular-Interval Time Series Data in a DSS File... 11-8
- 11.4 ZGIRTS – Get Irregular-Interval Time Series Data 11-12
- 11.5 ZPIRTS – Put Irregular-Interval Time Series Data 11-17
- 11.6 ZGTPFD – Get Paired Function Data 11-21
- 11.7 ZPTPFD – Put Paired Function Data 11-26
- 11.8 ZOPCAT – Open a Catalog File 11-32
- 11.9 ZCATLG – Catalog a DSS File 11-34
- 11.10 ZRDPN – Read Pathnames from a Catalog File by Reference Number 11-37

Appendices

Appendix A **Example Application**

Appendix B **Internal Subroutines**
- ZABORT – Abort Upon a Fatal Error B-1
- ZASSIG – Assign DSS File B-1
- ZBDUMP – Dump Buffers B-1
- ZBEGDT – Beginning Date B-1
- ZBKDAT – Block Data B-1
- ZCAOUT – Catalog Output B-2
- ZCATDR – Catalog Data Reference B-2
- ZCATFI – Catalog File B-2
- ZCATIT – Catalog Title B-2
- ZERROR – Fatal Error Processing B-2
- ZFSIZE – File Size B-2
- ZGETAD – Get Address B-3
- ZGETAG – Generate Tag B-3

Table of Contents (continued)

Appendices

Appendix B Internal Subroutines (continued)
ZGETCI – Get Compression Information B-3
ZGETRW – Get Record and Word B-3
ZGTAGS – Get Tag Scheme B-3
ZGTREC – Get a Disk Record B-3
ZHASH – Determine the Hash Code B-4
ZINCBK – Increment Block B-4
ZINIT – Initialize Variables B-4
ZIRBEG – Irregular Beginning Date B-4
ZIRDOW – Irregular Down B-4
ZLAHEY – Lahey® Open Adjustment B-4
ZMATCA – Match Catalog Pathname Parts B-5
ZMIN2R – Minutes to Real B-5
ZMOVBK – Move a Data Block B-5
ZMULTU – Multiple User B-5
ZNWBIN – New Bin B-5
ZNWRIT – New Write B-5
ZORDPN – Order Pathnames B-6
ZOWRIT – Old Write B-6
ZPRTC – Print Compression B-6
ZPTREC – Put a Disk Record B-6
ZR2MIN – Real to Minutes B-6
ZRDINF – Read Information Block B-6
ZRREC – Read Record B-7
ZRRTSB – Retrieve Regular-Interval Time Series Buffer B-7
ZSELCA – Selective Catalog B-7
ZSETCA – Set Catalog Parameters B-7
ZTAGFI – Tag File B-7
ZUPRTS – Update Regular-Interval Time Series B-7
ZWREC – Write Record B-8

Appendix C Data Screening Use of Data Flags

Appendix D Cross Reference Listing

Appendix E Abort Error Codes

Appendix F Summary of Subroutine Calling Sequences

1 Introduction

This manual is intended to provide programmers with information on how to interface programs with the Hydrologic Engineering Center's Data Storage System (HECDSS or DSS), and to provide a background on DSS and its capabilities. This document is intended to be used both as an introduction to programming with DSS and as a reference manual.

DSS is written in FORTRAN77 and is designed to be called by FORTRAN programs. It is assumed that the reader has a working knowledge of FORTRAN. The DSS makes use of several subroutines in the software library "HECLIB". The HECLIB programmer's manual is a companion document that should be accessible when programming with DSS.

1.1 Background

HEC developed the Data Storage System (or DSS) to meet needs for data storage and retrieval for water resource studies. The system, which has been under development since 1979, enables efficient storage and retrieval of time series and other data-types for which storage in blocks of contiguous data elements is most appropriate. The DSS consists of a library of subroutines that can be readily used with applications programs to enable retrieval and storage of information. At present approximately twenty applications programs have been adapted in this fashion, including the widely used program Flood Hydrograph Package (HEC-1) and the Expected Annual Damage (EAD) program. In addition, approximately seventeen DSS utility programs have been developed. A number of these programs are for data entry, for example from the U.S. Geological Survey WATSTORE database, or from National Weather Service precipitation data files. Other utility programs include a powerful graphics program, a report generator, and a program for performing mathematical transformations. Macros, selection screens, and other user interface features combine with DSS products to provide a set of tools whose application is limited only by the ingenuity of the user.

The DSS was the outgrowth of a need that emerged in the mid 1970's. During that time most studies were performed in a step-wise fashion, passing data from one analysis program to another in a manual mode. While this was functional, it was not very productive. Programs that used the same type of data, or that were sequentially related, did not use a common data format. Also, this required that each program have its own set of graphics routines, or other such functions, to aid in the program's use.

The Kissimmee River study performed by the HEC for the Jacksonville District beginning in 1978 required that an orderly approach be used to properly manage the study data and the analysis results. A large number of alternative plans and conditions were to be processed in this project. This study gave birth to the first version of DSS. The basic design provided for the storage of data in a standard form, independent of any particular program. The data would be provided to the programs when it was needed, and results would be stored in the same independent form for use by utilities and other applications programs. The early design of DSS was conceived to support files containing many hundreds of data records, or even as many as a few thousand. As the use of DSS expanded into real-time data storage applications, data files were written to manage as many as 10,000 to 20,000 records. The current DSS version is now designed for rapid storage and retrieval of files containing as few as forty to fity records, or as many as 100,000 or more.

1.2 DSS Contrasted with Other Database Systems

DSS is designed for the storage and retrieval of large sets, or series, of data. This includes daily flow values, hourly precipitation measurements, rating tables, and pages of text information. DSS is the least efficient with small sets of data, or elements, the type for which most commercial database systems are designed. For example, such elemental data might include employee records, accounting data, and inventory of stock.

DSS incorporates a modified hashing algorithm and hierarchical design for database accesses. This algorithm provides quick access to data sets and an efficient means of adding new data sets to the database. Commercial databases usually employ a relational model. In this model data is stored in a related manner, so that the database can be viewed essentially as a collection of tables (composed of rows and columns). This type of system requires the construction of a data definition file. Although a relational database requires some initial setup, it can effectively store short data sets comprised of both characters and numbers. The structure usually provides a advantageous method of retrieving and storing data based on the data definition. Most of the newer systems use the ANSI ratified Structured Query Language (SQL) to access data. (An example of SQL might be "SELECT models FROM parts WHERE model=3210".)

While relational databases are ideal for elemental data sets, they are not as practical for longer series of data. DSS, however, is designed for such sets of data. A data definition file like relational models require does not define DSS database files; thus there is no set up required by the user. DSS data is defined by the pathname and conventions used. The type of data generally stored in DSS does not lend itself well to a query language such as SQL (although the selective catalog feature has some similar capabilities). Also, the DSS is made up of a set of utility programs and application programs, whereas commercial databases are typically accessed by one main program.

1.3 General Information

Direct access files are used for DSS data storage. The user conventionally names these files. Such a file cannot be directly viewed or accessed by programs that do not interface to the DSS. A DSS file does not need to exist before the first DSS subroutine (ZOPEN) is called; it will be created if it is not present. Nor are DSS files initialized or set up prior to use.

Data within a database file is stored in blocks, or records, and a unique identifier called a "pathname" identifies each record. Each time data is stored or retrieved; its pathname must be given. Along with the data, information about the data (e.g., units of the data) is stored in an internal "header". The DSS automatically stores the name of the program writing the data, the number of times the data has been written, and the last write date and time.

Through information contained in the pathname and internal header, the data is self-documented. That is, no additional information is required to identify it. This feature of the database allows information to be recognized and understood years after it was stored.

Data of most any type, using a pathname of any structure (up to eighty characters), can be stored by DSS. To facilitate the ability of application and utility programs to work with and display data, standard record conventions were developed. These conventions define what should be contained in a pathname, how data is stored and what additional information is stored along with the data. For regular-interval time series data (e.g., hourly data), the conventions specify that data are stored in blocks of a standard length, uniform for that time interval, with a pathname that contains the date of the beginning of the block and the time interval. The conventions identify how a pathname for that data should be constructed. Conventions have been defined for regular and irregular interval time series data, paired (curve) data, and text (alphanumeric) data. Conventions for other types of data have been proposed.

A major component of the DSS conventions is the structure of the pathname. All conventions segment the pathname into six parts. Parts are referenced by the letters A, B, C, D, E, and F, and delimited by a slash "/", as follows:

/A/B/C/D/E/F/

An example pathname that follows the time-series convention is:

/ALLEGHENY/KINZUA/FLOW-RES IN/01JAN1972/1DAY/OBS/

A non-standard convention may be used to store and retrieve data from a DSS database file by applications programs for which it is defined. However DSS utility programs (DSSUTL and DSPLAY) will have only limited to the data. Such data can be copied, tabulated, renamed, etc. It cannot be edited or graphically displayed. The non-standard convention does not have to have a segmented pathname as shown above, although it is recommended. A pathname that does not follow the standard conventions might be:

/DATA SET 5-A/

Although the beginning and ending slash in this pathname is not required, it is highly recommended because the slashes identify the string as a pathname, and allow limited use of the record in utility programs. A preferable modification of this pathname would be:

/DATA SET 5-A//////

In order to maximize the effectiveness of a database, several users should be able to access the same database file at the same time. For example, a flood forecast model might need to retrieve data from the file at the same time data elsewhere in the file is being updated. To do this, the database must incorporate a method of handling multiple users of the same file at virtually the same time. A DSS multiple user scheme accomplishes this by a first come - first served approach. When a program requests to write to the file, the DSS software will queue that request and hold it until all prior requests are completed. Typical delays take only a fraction of a second and are not detectable by users. This feature does not require any added programming; the DSS accomplishes multiple user access internally.

A DSS version identifier is stored internally along with each DSS file. The version is in the form "6-FG", where the number portion (e.g., 6) indicates the file structure, the first letter signifies moderate software modifications or a collection of minor changes, and the last letter connotes minor changes. DSS software is not compatible with files that were created with a different number version (e.g., a file version of 4 is not compatible with software with a version of five). (On HARRIS "H series" computers, DSS Version 6 software will access DSS Version 4 files.) The DSS version is printed in the ZOPEN statement and in the catalog file.

1.4 A Short Description of How DSS Works

DSS Version 6 utilizes a modified hash algorithm based upon the pathname to store and retrieve data. This structure allows the DSS to "jump" to the location of the data in the file based upon a disk address determined from the pathname's hash code. The DSS does its own blocking, providing the maximum use of disk space while allowing varying lengths for data records.

The first portion of a DSS file is the "permanent section" of the file. This area contains information about the DSS file, such as its size, how many records are contained in it, and the amount of inactive space. Following the permanent section is the "hash table". When retrieving data, the DSS computes a hash code from the pathname. This code points to a location in the hash table, which contains the disk address of the record's "pathname bin". The pathname bin contains pathnames that have the same hash code and the disk addresses of their data sets. If there are more pathnames with the same hash code than that which will fit in the bin (typically five), an "overflow bin" is used for the remainder of the pathnames. The hash code size is usually adjusted so there is few overflow bins. This structure allows most data to be retrieved in three or four disk accesses. These accesses are to the hash table, the pathname bin (and possibly an overflow bin), and the data record.

The process for storing data operates in a similar same manner. First it is determined if the data record already exists. If it does, then the existing data area is written over with the new data (unless the new data record size is larger). If the record does not exist, then the data is appended to the end of the file, and the appropriate address and information are added to the hash table and pathname bin. The address buffers are structured in such a way that if an abort occurs (e.g., a system crash) during a write operation, only the data record being written will be lost.

The structure described above is for a "dynamic" hash table. It provides for a wide range in the number of records in a DSS file while efficiently balancing disk space and record access times. A variation in this algorithm is for "stable" files. In a stable file, a hash table is not utilized. Instead, the pathname hash code indicates directly the location of the pathname bin without having to access an intermediate table. In this type of file, all pathname bins are pre-allocated when the file is first created. A stable type file saves one disk access for each record, but it causes the DSS files to be large when they are first opened and can be less efficient with disk space, especially when there is less than the optimum number of records in the file. This method is intended for somewhat stable databases that do not frequently change in size (e.g., a master database file).

The hash code size can be adjusted for a new database to optimize storage and retrieval of data according to the expected number of records in the file. It should be noted that any of the sizes will operate with any number of records, but an incorrect size will not be as efficient as the appropriate one. When a user squeezes a database file with the program DSSUTL, the size parameter is automatically adjusted based on the number of records in the file at that time. The possible sizes are:

Size Name	**Hash Size**	**Ideal Number of Records**	**Target Range of Records**
Tiny	8	20	1 - 50
Extra-Small	32	50	1 - 200
Small	128	200	100 - 1,000
Medium (default)	512	1,000	200 - 5,000
Large	1024	4,000	1,000 - 10,000
Extra-Large	2048	10,000	2,000 - 20,000
Huge	4096	20,000	5,000 - 50,000
Extra-Huge	8192	50,000	>25,000

The experienced user who desires control over the optimization of database files generally sets the table type and size. The default parameters are sufficient for most users. These parameters may be set by a call to ZSET prior to opening a new file with ZOPEN, or by the "SQUEEZE" or "OPEN" commands in DSSUTL.

The DSS software has been tested with database files containing over 200,000 records on MS-DOS and other computers. With larger files, use of the catalog becomes impracticable and all references to records should be made with their pathnames only. (When a catalog reference number is used, a sequential search is made through the catalog file for the pathname corresponding to that number. This search can take a long time for large catalogs.) When a correct table size is selected, the DSS shows little degradation in storing or retrieving records from a very large file. However, such large files are not recommended because of general computer limitations, such as the backup and copying of those files.

1.5 Programming with DSS

1.5.1 Opening and Closing DSS Files

DSS was designed to be consistent with the style set by the FORTRAN77 standards. Before data can be accessed in a DSS database file, the file must first be opened with subroutine ZOPEN. After all transactions are completed, the file is closed with subroutine ZCLOSE. Any calls to store or retrieve data without the file opened by ZOPEN will result in an error. Generally, a DSS file should not be opened more than once during the execution of a program. Never open or close a DSS file using a FORTRAN OPEN or CLOSE statement; DSS files must only be opened by ZOPEN and closed by ZCLOSE.

Programs can access more than one DSS file simultaneously. For example, data can be retrieved from one DSS file and stored in another. The first file opened by DSS will be connected to Unit 71, and then the second will be connected to Unit 72, etc. Calling ZSET prior to ZOPEN can modify the unit number. A separate IFLTAB array is needed for each opened DSS file.

1.5.2 The IFLTAB Array

Programs accessing DSS must provide an array named IFLTAB for each DSS file opened. This array holds file pointers and parameters, and it can be viewed in a similar manner to a FORTRAN unit number. IFLTAB is an integer array that is dimensioned to 300 INTEGER*4 words or 600 INTEGER*2 words. On HARRIS "H series" computers, where DSS Version 4 files might be accessed, the array must be dimensioned to 1200 INTEGER*3 words. The IFLTAB array must not be altered after the DSS file has been opened. A separate IFLTAB array is needed for each DSS file opened concurrently.

Certain elements in IFLTAB contain key flags that are checked frequently by the DSS. If one of these flags has changed, then memory has been overwritten and the array is corrupted. When this occurs an error message is printed and the program is aborted. If this error occurs, it must be corrected before further work is done (bounds checking is useful for this).

1.5.3 DSS Subroutines

Higher-level subroutines are available to store and retrieve time series data, paired (curve) data, and text data. Time series data is composed of two categories: regular-interval and irregular-interval data. For regular-interval time series data, the date and time of each data value is implied by its position within the record. For irregular-interval time series data, each data value has its own date and time stamp associated with it. Regular interval time-series data is stored by subroutine ZSRTS or ZSRTSX and is retrieved by ZRRTS or ZRRTSX. Irregular interval time-series data is stored by subroutine ZSITS or ZSITSX and is retrieved by ZRITS or ZRITSX. Paired data, which usually defines a curve or set of curves, is stored by ZSPD and is retrieved by ZRPD. Text data may be stored by subroutine ZSTEXT or ZSTXTA and retrieved by ZRTEXT or ZRTXTA. Data that does not meet any of the conventions established may be stored in a DSS file by calling subroutine ZWRITE and retrieved by calling ZREAD. These subroutines should only be used when no other subroutines are available for the type of data being used.

Other subroutines are available for the following tasks: to set parameters, such as the program name (ZSET); to inquire about the value of parameters (ZINQIR); to generate a pathname (ZPATH); to get pathname parts from a input line (ZGPNP); to break a pathname into separate parts (ZUPATH); to determine if a record exists and what its data type is (ZDTYPE); and to provide cataloging and other utility functions (e.g., ZCAT).

1.5.4 Message Control

DSS messages are written to FORTRAN Unit 6. This message unit number may be set to some other number by calling the subroutine ZSET. The amount of message output can be controlled by setting "MLEVEL" (the message level) with ZSET. This message level can vary from zero to fifteen. Level zero will cause information to be printed only when a sever error occurs. Most of the higher-level subroutines, such as ZSRTS, will print an internal trace for debugging when the message level is set to nine. Message levels higher than nine are for installing DSS on a new computer and generally do not provide any additional information for the programmer. The default level is four, which prints the pathname whenever a record is retrieved or stored.

1.6 Typical Order of Calling DSS Subroutines

A typical sequence of programming instructions used to store or retrieve data in a DSS file is as follows:

1) The program detects that a DSS database file is to be accessed. For several HEC application programs a "ZR" or "ZW" record in the input triggers this.
2) The DSS file is opened by ZOPEN. The file name is often obtained from the execution line or from the input. If ZOPEN is called in a routine that may be called several times, a logical flag may be set to indicate that the file has already been opened (so the file will not be opened a second time). If data values are to be stored, the program name is set by a call to ZSET with the 'PROGRAM' parameter.
3) A DSS pathname for the data to be accessed is constructed from information on the "ZR" or "ZW" record. Often the subroutine ZGPNP (Get Pathname Parts) and ZPATH (Form Pathname) are called to produce the pathname.
4) If data values are to be stored, they are organized sequentially into an array. The data units and type are identified. If the data is time series, the associated times are identified.
5) The DSS storage or retrieval subroutine is called (e.g., ZSRTS or ZRRTS).
6) The status of the call is checked. If an error occurred, the appropriate action is taken.
7) If data was retrieved, it is used as input for the program.
8) The remaining sets of data are retrieved or stored, until all DSS accesses are completed.
9) When all DSS accesses are complete, the file is closed by ZCLOSE (often this is called at the end of the program).

1.7 Machine Specifics

1.7.1 HARRIS Computers

Access to DSS subroutines is accomplished by linking with the library HECLIB. HECLIB is stored in the qualifier 2000SYSS. These subroutines are compiled without any compiler options. A typical compilation and linking procedure is as follows:

```
SAUF77 MYSOURCE
VU.R MYPROG
LIB 2000SYSS*HECLIB *LIBERY
BEGIN
```

The IFLTAB array must be dimensioned to 1200 INTEGER*3 words on the Harris, to allow compatibility with DSS version 4 files. Unit 6 must be assigned to standard output prior to calling DSS, unless the message unit is reset with subroutine ZSET.

It should be noted that an old FORTRAN 66 version of HECLIB exists on some Harris computers. This library does not contain the subroutines documented in this manual.

1.7.2 Microcomputers Using Microsoft® FORTRAN

The DSS (and HECLIB) subroutines are compiled with Microsoft® FORTRAN using a default word length of INTEGER*2. All integer arguments passed to DSS subroutines must be this length, unless otherwise indicated. The Julian dates and the time interval passed to the time series routines are some of the variables that must have a word length of INTEGER*4. The IFLTAB array should be dimensioned to 600 INTEGER*2 words or 300 INTEGER*4 words.

The DSS subroutines are distributed in the library \LIB\HECLIB.LIB. At the time of the publication of this manual, the subroutines were compiled with Microsoft® FORTRAN Version 5.0. The following options were used to compile the library:

```
/Zl /FPi /Ols /4I2 /Gt80 /c
```

The Microsoft® linker supplied with the FORTRAN compiler should be used to link your program. The linker supplied with DOS should not be used, as it does not know about the FORTRAN libraries (both linkers are named "LINK"). Typically, programs accessing DSS require additional segments. The number of segments can be increased by the link option /SE:number-of-segments. Generally 400 to 500 segments are needed. Occasionally the stack size may also need to be increased. The instructions necessary to compile and link a program with DSS might be the following:

```
FL /FPi /4I2 /c MYPROG.FOR
LINK MYPROG /SE:400 /ST:3000 /NOD /E
Library:  HECLIB+LLIBFORE
```

(The DSS utility programs are linked with the large memory FORTRAN library with math chip emulation software.)

Occasionally, large programs will require a larger environment size. The error "HEAP SPACE EXCEEDED", obtained when executing the program, often indicates an insufficient environment size. To increase the size, the /P and /E switches need to be added to the COMMAND.COM command in your CONFIG.SYS file. An example of this is:

```
shell=c:\dos\command.com /p/e:512
```

More information about this may be found in your DOS manual under the COMMAND command.

DSS files generated with Microsoft® FORTRAN are binary compatible with software compiled with different compilers for DOS and with most UNIX computers. A DSS file can be copied (binary) to a UNIX computer, then used by DSS programs on that computer (or visa versa). Software linked to the HECLIB produced after February 1991 contains file locking features that allow multiple user access of DSS files on a networked system.

1.7.3 Microcomputers Using Lahey® FORTRAN

The DSS and HECLIB subroutines have been compiled with Lahey® and Lahey® extended memory (32 bit) FORTRAN. The library compiled with the regular Lahey® compiler has a default integer word length of INTEGER*2. The library compiled with extended memory Lahey® has a default integer word length of INTEGER*4. Because not all HECLIB subroutines have been converted to Lahey®, it is preferred that the Microsoft® FORTRAN HECLIB be used, if possible. The IFLTAB array must be dimensioned to 600 INTEGER*2 words or 300 INTEGER*4 words.

The DSS library compiled with regular Lahey® FORTRAN is distributed in the file \LIB\HECLIBL.LIB, and the library compiled with extended memory Lahey® is distributed in \LIB\HECLIBL3.LIB. The following options are used in the F77L3.FIG file to compile the library with extended memory Lahey®:

```
/n0/n7/B/C/D/nF/nH/I/nK/nL/nO/P/nQ1/R/nS/nT/W/nX/Z1
```

Note that the /D option is needed so that DSS files (which are direct access) will not have a Lahey® header. The subroutines compiled with the regular Lahey® compiler use the option "/T" to default integer word lengths to INTEGER*2.

DSS files generated with either Lahey® compiler are binary compatible with DSS files created with the Microsoft compiler and with most UNIX computers. However, the Lahey® ZOPEN may adjust the DSS file size if it has been opened by program compiled by Microsoft®.

1.7.4 UNIX Operating Systems

The DSS and HECLIB subroutines have been compiled in several UNIX operating systems. (Contact HEC for a current list of which UNIX computers for which DSS is available.) The library is compiled with a default integer word length of INTEGER*4. The IFLTAB array should be dimensioned to 300 INTEGER*4 words.

Generally, the library is compiled using the default compiler options. The library is distributed in the file:

```
/usr/hec/lib/heclib.a
```

DSS files are generally compatible across UNIX and DOS computers. However, DSS files created by a library not converted by HEC may be incompatible with other computers.

2 General Subroutines

This chapter describes general DSS subroutines, including ZOPEN and ZCLOSE, both of which must be called by all programs accessing DSS. ZOPEN opens (or connects) a DSS file before any data transactions can occur. ZCLOSE closes (or disconnects) a DSS file after all accesses to that file are complete. ZOPEN and ZCLOSE should be called once for each DSS file accessed.

ZFNAME adds any default extensions to a DSS file name (e.g., ".dss") and determines if that file exists. ZFVER will determine the DSS version number of a file before it is opened with ZOPEN.

ZDTYPE determines if a record exists and, if it does, returns its data type (e.g., whether it is regular-interval time series, paired data, etc.).

ZSET provides a means of setting several DSS parameters during execution. Items, such as the program name, and the message level (trace) may be set. ZINQIR provides a means of determining what parameters are set to. This includes items such as the message level, the DSS version, and the number of records in a file.

2.1 ZOPEN - Open a DSS File

Purpose:

ZOPEN is used to open (or connect) a DSS file. If the file does not exist, ZOPEN will create it with public access. Except for the subroutines ZFNAME, ZFVER and ZSET, ZOPEN must be called prior to any other DSS subroutine.

ZOPEN must be called once (and only once unless the file is closed) for each DSS file to be accessed. DSS files cannot be opened or connected by any other means.

Calling Sequence:

```
CALL ZOPEN (IFLTAB, CNAME, IOSTAT)
```

Declarations:

```
INTEGER IFLTAB(600), IOSTAT
CHARACTER CNAME*(*)
```

Argument Description:

IFLTAB	Input/ Output	IFLTAB is an array used by the DSS software to manage the file. After the DSS file has been opened, that file is referred to in DSS subroutines by IFLTAB. Each DSS file opened must have its own IFLTAB array, and that array must not be altered. See **Remarks** section concerning the required length of IFLTAB.
CNAME	Input/ Output	The name of the DSS file to be opened. If the computer uses file name extensions (e.g., "db.dss"), ZOPEN will append the default extension (".dss") to the name if it has none.
IOSTAT	Output	A status parameter indicating the success of the operation. If IOSTAT is returned with zero, then the file was opened successfully. If IOSTAT is returned non-zero, then a fatal error occurred, and the file was not opened. Do not attempt to retrieve or store data if IOSTAT is non-zero. The possible values are:

IOSTAT	**Description**
0	Successful open.
-1	Unable to create the DSS file.
-2	Unable to connect to the file.
-3	Incompatible DSS versions.
-10	No filename was provided.

IOSTAT	Description
>0	Unable to OPEN the file. See the IOSTAT parameter for the OPEN statement of your FORTRAN manual.

Remarks:

For DSS Version 6, IFLTAB must have a length of 300 long integer words, or 600 short integer words. On HARRIS computers, its length must be 1200 (short) integer words for compatibility with version 4 of DSS. Hereafter, its length will be shown as 600 (short) integer words.

ZOPEN will create the DSS file if it does not exist. Subroutine ZFNAME can be called prior to ZOPEN to determine if the file exists.

ZOPEN must be called once per DSS file used. ZCLOSE must be called when all references to that file are complete. After ZOPEN, a DSS file is referred to by use of the IFLTAB array. By default, the first DSS file opened will be connected to Unit 71, the second to Unit 72, etc. This parameter may be changed by a call to ZSET.

Do not attempt to open a DSS file with an OPEN statement, or ASSIGN or ATTACH a DSS file, as ZOPEN will accomplish this. To do so may destroy the DSS file (IBM mainframes are an exception).

The DSS version of a file (e.g., 5-BD or 6-FA) can be determined prior to calling ZOPEN with subroutine ZFVER. Once the file has been opened, a call to ZINQIR will obtain the version.

Example:

```
      INTEGER IFLTAB(600)
      CHARACTER CNAME*64, CNNAME*64
      LOGICAL LEXIST
C
C     Connect unit 6 to the standard output via ATTACH.
      CALL ATTACH ( 6, 'OUTPUT', 'STDOUT', ' ', CNAME, ISTAT)
C     Get the name of the DSS file from the execution line via ATTACH.
      CALL ATTACH ( IDUM, 'DSSFILE', ' ', 'NOP', CNAME, ISTAT)
      CALL ATTEND
C
C     Because data is to be retrieved, make sure the DSS file exists
C     before calling ZOPEN (as ZOPEN would create it).
      CALL ZFNAME (CNAME, CNNAME, NNAME, LEXIST)
      IF (.NOT.LEXIST) THEN
         WRITE (6,10) CNNAME
 10      FORMAT (' ** The DSS File does not exist: ',A)
```

```
         GO TO 900
      ENDIF
C
      CALL ZOPEN (IFLTAB, CNAME, IOSTAT)
      IF (IOSTAT.NE.0) THEN
         WRITE (6,20) IOSTAT, CNAME
20       FORMAT (' *** Error in opening DSS file, status:',I5,
     *   ',  Name: ',A)
         GO TO 900
      ENDIF
```

2.2 ZCLOSE - Close a DSS File

Purpose:

ZCLOSE is used to close (or disconnect) a DSS file after all DSS transactions with that file have been completed. ZCLOSE must be called once (and only once) for each DSS file opened.

Calling Sequence:

```
CALL ZCLOSE (IFLTAB)
```

Declarations:

```
INTEGER IFLTAB(600)
```

Argument Description:

IFLTAB	Input/ Output	The workspace used by DSS to manage the file. This is the same array that is used in the ZOPEN call and most other DSS subroutines.

Remarks:

ZCLOSE "zeros" the IFLTAB array so that another DSS file can be opened with it afterwards. Once a file is closed, it must be reopened before it can be accessed again. Information about the file size and number of records is written to standard output by ZCLOSE.

2.3 ZFNAME - Add Filename Extension and Determine if the File Exists

Purpose:

On computers where file name extensions are used, ZFNAME will add the default extensions to a DSS file name (unless the file name already has an extension) and determine if that file exists. File name extensions include ".DSS" on MS-DOS computers and ".dss" on Unix machines. ZFNAME is an optional subroutine, as ZOPEN will automatically add extensions to the name.

Calling Sequence:

```
CALL ZFNAME (CNAMIN, CNAME, NNAME, LEXIST)
```

Declarations:

```
CHARACTER CNAMIN*(*), CNAME*64
INTEGER NNAME
LOGICAL LEXIST
```

Argument Description:

CNAMIN	Input	The DSS filename. This is generally the name a user enters. The name may or may not have extensions.
CNAME	Output	The DSS filename with the default extensions added to it. If the file name already has an extension, or the computer does not use extensions, no extension will be added.
NNAME	Output	The number of non-blank characters in CNAME (i.e., its length).
LEXIST	Output	A logical variable set to .TRUE. if the file exists, or .FALSE. if it does not.

Remarks:

An extension in the name is defined as a period and any characters that follow it. On MS-DOS computers the default extension is ".DSS". However, a DSS file does not necessarily have to have that extension (although DSS utility programs will not "highlight" files that do not have the ".DSS" extension). Any extension or no extensions are legal for a DSS file name, as long as a period appears in the name. For example, "MYDB.DAT", and "MYDB." are allowable names, whereas "MYDB" (with no period) will have ".DSS" automatically added to it. The default extension of the catalog file is ".DSC".

Similarly, UNIX computers use a default extension of ".dss", and the catalog has an extension of ".dssc". No extension will be added if the file name contains a period within it. On computers where extensions are not commonly used (e.g., HARRIS), ZFNAME will not change the name (CNAME will equal CNAMIN), and will only determine if the file exists.

Example:

```
C     Open a DSS file for retrieving data.
C
      CHARACTER CNAMIN*64, CNAME*64
      LOGICAL LEXIST
C
      CALL ATTACH ( 0, 'DSS', ' ', 'NOP', CNAMIN, ISTAT)
      . . .
C
      IF (CNAMIN(1:1).EQ.' ') THEN
         WRITE (6,*)'No DSS file name provided!'
         GO TO 900
      ENDIF
C
      CALL ZFNAME (CNAMIN, CNAME, NNAME, LEXIST)
C
      IF (.NOT.LEXIST) THEN
         WRITE (6,20) CNAME(1:NNAME)
20       FORMAT (' The DSS File ',A,' Does Not Exist!',/
     *      ' The DSS file must exist in order to retrieve data.')
         GO TO 900
      ENDIF
C
      CALL ZOPEN (IFLTAB, CNAME, ISTAT)
C
```

2.4 ZDTYPE - Determine a Record's Data Type and if it Exists

Purpose:

ZDTYPE determines if a record exists and, if it does, returns its data type (e.g., whether it is regular-interval time series, paired data, etc.). If the record does not exist, ZDTYPE will examine the pathname in an attempt to determine if it follows the time-series conventions.

Calling Sequence:

```
CALL ZDTYPE (IFLTAB, CPATH, NSIZE, LEXIST, CDTYPE, IDTYPE)
```

Declarations:

```
INTEGER IFLTAB(600), NSIZE, IDTYPE
CHARACTER CPATH*80, CDTYPE*3
LOGICAL LEXIST
```

Argument Description:

IFLTAB	Input/ Output	The DSS workspace used to manage the DSS file. This is the same array used in the ZOPEN call.
CPATH	Input	The pathname of the record to examine. The length of CPATH is implicit (e.g., CPATH(1:NPATH)).
NSIZE	Output	The size of the data portion of the record, given relative to floating point values (i.e., if the record consisted of real numbers). If the record does not exist, NSIZE is returned as zero.
LEXIST	Output	logical variable set to .TRUE. if the record exists.
CDTYPE	Output	The character representation of the data type. This variable corresponds to the data type displayed in the catalog file and as shown below.
IDTYPE	Output	An integer flag indicating the data type. The currently defined data types are as follows:

IDTYPE	CDTYPE	Data Type
0	UND	Undefined
100	RTS	Regular-Interval Time Series Data
110	ITS	Irregular-Interval Time Series Data
200	PD	Paired Data
300	TXT	Text Data

Remarks:

ZDTYPE replaces the function of subroutine ZCHECK. The data type is stored by the standard storage routines (e.g., ZSRTS), while ZWRITE sets the data type to 0 (undefined). Data types cannot be changed or set externally.

Example:

```
C     Tabulate different types of data in a DSS file.
C
      NTEGER IFLTAB(600)
      CHARACTER CPATH*80, CDTYPE*3
      LOGICAL LEXIST
C     . . .
C
C     Open the DSS file.
      CALL ZOPEN (IFLTAB, ...
C
C     Get the pathname.
      WRITE (6,*) 'Enter DSS Pathname'
      READ (5, 20, END=800, ERR=900) CPATH
 20   FORMAT (A)
C
C     Determine if the record exists, and its data type.
      CALL ZDTYPE (IFLTAB, CPATH, NSIZE, LEXIST, CDTYPE, IDTYPE)
C
C     Print a message if the record does not exist, and it
C     is not time series.  (If the data is time series with a time
C     window, the "D" (date) part is not required; thus the record
C     may exist with another D part.)
      IF ((.NOT.LEXIST).AND.(CDTYPE(2:3).NE.'TS')) THEN
         WRITE (6,40) CPATH
 40      FORMAT (' Record Does Not Exist: ',A)
         GO TO 200
      ENDIF
C
      IF (IDTYPE.EQ.100) THEN
         CALL ZRRTS ( ...
C
      ELSE IF (IDTYPE.EQ.110) THEN
         CALL ZRITS ( ...
C
      ELSE IF (IDTYPE.EQ.200) THEN
         CALL ZRPD ( ...
C
      ELSE IF (IDTYPE.EQ.300) THEN
         CALL ZRTEXT ( ...
C
      ELSE
         CALL ZREAD ( ...
C
      ENDIF
```

2.5 ZSET - Set DSS Parameters

Purpose:

ZSET provides a means of resetting several default parameters used by DSS. This includes items such as the name of the program storing data, the DSS file unit number to use, etc. ZSET may be called at any time (before or after ZOPEN).

Calling Sequence:

```
CALL ZSET (CITEM, CSTR, INUMB)
```

Declarations:

```
INTEGER INUMB
CHARACTER CITEM*4, CSTR*6
```

Argument Description:

CITEM	Input	The item to be set. Items may be abbreviated to the first four characters of the name. A list of the available items follows (see Summary Table, page 2-12).
CSTR	Input	A character string containing the value to be set. If the parameter to be set is an integer number, this argument is ignored (and can be ' ').
INUMB	Input	The integer number containing the value to be set. If the parameter to be set is a character string, this argument is ignored.

Remarks:

The precision and tolerance parameters are completely independent of each other, as are the size and table type parameters. The size parameter is set more frequently than the table type. The message unit (MUNIT) must be connected to a terminal or console to set the catalog status or squeeze status. If a file has system read only permission, READONLY is set to ON automatically by ZOPEN. The exclusive use mode may not be available on all computers. The exclusive write lock mode should be set sparingly.

Summary Table:

		Default	
CITEM	**Description**	**CSTR**	**INUM**
'PROGRAM'	Sets the name of the program to store with the data.	'Undefi'	
'UNIT'	Sets the unit number of the next DSS file to be opened (via ZOPEN).		71
'MLEVEL'	Sets the message level (trace) for MUNIT.		4
'MUNIT'	Sets the unit number of the message output (standard out).		6
'80COL'	Abbreviates the size of messages to fit within 80 columns.	'OFF'	
'TAG'	Sets the tag of the next record to be written.		
'PRECISION'	Sets the precision of the data for use by utility programs.		0
'TOLERANCE'	Sets the tolerance of regular-interval times series data to prevent overwriting unchanged data.		0
'PROTECT'	Protects existing records from being written over.	'OFF'	
'READONLY'	Places the file in a "read only" mode.	'OFF'	
'EXCLUSIVE'	Places the file in an "exclusive use" mode.	'OFF	
'WLOCK'	Places the file in an "exclusive write lock" mode.	'OFF'	
'SIZE'	Sets the internal hash table size for a new file.	'MEDIUM'	1000
'TABLE'	Indicates whether a dynamic or stable hash table should be used for a new file.	'DYNAMIC'	
'COMP'	Re-compresses regular-interval time series data when ZCOFIL is called.	'OFF'	
'CAST'	Catalog Status - Causes a status line to be displayed during a catalog.	'OFF'	
'SQST'	Squeeze Status - Causes a status line to be displayed during a file copy.	'OFF'	
'MAP'	Causes a catalog map to be created when a new catalog is generated.	'OFF'	
'MAPUNIT'	Sets the unit number to use for the catalog map file.		

Parameters

PROGRAM	This sets the name of the program, which is stored with the data. The name can be up to six characters long. This call should be made prior to calling any DSS storage subroutines. The default value is 'Undefi'.

UNIT	This sets the unit number of the next DSS file to open. In order to use this parameter, ZSET must be called prior to ZOPEN for that file (you cannot change the unit number once the file has been opened). This setting will not change the unit number of other DSS files to be opened. The default unit number of the first DSS file opened is 71, the second is 72, and for subsequent files the unit number is incremented by one.
MLEVEL	This sets the level of messages to be written to the output unit (MUNIT). The level ranges from "abort" only messages to internal trace messages. Level 4 is the default. Levels greater than five provide debugging messages for some DSS subroutines (see documentation on the specific subroutine for debug levels). Level ten and above are used for first time installation of DSS on a new computer (and will generate several pages of cryptic trace for one write). A higher level incorporates all lower level messages.

Level	**Type of Messages**
0	Messages from an abort only.
1	ZOPEN and ZCLOSE statements.
2	Error and warning messages.
3	ZWRITE messages.
4	ZREAD messages (default).
7	Beginning level of debugging messages.
8	Intermediate level of debugging messages.
9	Maximum level of debugging messages.
>10	Internal DSS trace (don't use unless you know what you are doing).

MUNIT	This sets the unit number for messages (the standard output). The default is Unit 6. The unit must be opened prior to calling ZSET.
80COL	When CSTR is 'ON', DSS output messages (e.g., ZWRITE and ZREAD messages) are abbreviated so that they will (usually) fit within eighty columns. If pathnames are long, the eighty-column size may be exceeded.
TAG	This causes CSTR to be the tag for the next new record written (or multiple records for a single call to store time-series data). The tag can be up to eight characters long. It must begin with a non-numeric character and cannot have embedded blanks or commas. See Chapter 7 for more information on tags. Default tags will be used for subsequent writes (ZSET must be called prior to every write where a tag is to be set).
PRECISION	This stores a precision value to be used by utility programs when tabulating the record. The precision is a number between one and seven that represents the minimum number of places to the right of the decimal

	that must be displayed. If the precision is two (2), the data will be displayed to the nearest hundredth; three (3) indicates to the nearest thousandth. A zero indicates no precision value is set. The precision is set only for the next record written (i.e., call ZSET just prior to each write). Subsequent writes will not store a precision value unless ZSET is called again.
TOLERANCE	When storing regular-interval time series data (ZSRTS), this setting provides a means of preventing existing data from being overwritten with the same data but at a possibly lesser precision. This is designed to preserve the precision of unchanged data during editing. The tolerance is a number between zero and seven that indicates the accuracy of data by the number of places to the right of the decimal. For example, if a tolerance of two is set, a data value that was within a hundredth of the value currently stored would not replace that value (138.76 would not replace 138.7574, but 138.77 would). The tolerance is set only for the next record written (i.e., call ZSET just prior to each ZSRTS). No tolerance is checked for subsequent writes (unless ZSET is called again). This setting is ignored for new records.
PROTECT	When CSTR is 'ON', this prevents the next record to be written from writing over an existing record (with the same pathname). The protection is only for the next record written (i.e., call ZSET just prior to each write). Existing records for subsequent writes are not protected (unless ZSET is called again).
READONLY	When CSTR is 'ON' prior to the call to ZOPEN, this will cause the next file opened to be placed in a "read only" mode. Nothing can be written to the file in this mode. This setting does not prevent other programs from writing to the file. This flag applies only to the next file opened; subsequent files opened will be in a read/write mode unless ZSET is called again.
EXCLUSIVE	When CSTR is 'ON' prior to a call to ZOPEN, this will cause the next file opened to be placed in a "exclusive use" mode. No other programs can access the file in this mode. This flag applies only to the next file opened; subsequent files opened will be opened normally unless ZSET is called again.
WLOCK	When CSTR is 'ON' prior to the call to ZOPEN, this will cause the next file opened to be placed in a "exclusive write lock" mode. In this mode the file is placed in an exclusive use mode, and portions of the main address tables are kept in memory, causing writing to be slightly faster. If the file is not correctly closed (a system crash or power failure), the file will have to be squeezed by DSSUTL prior to being used. No other programs can access the file in this mode. This flag applies only to the

	next file opened; subsequent files will be opened normally unless ZSET is called again.
SIZE	This parameter sets the size of the internal hash-address table (according to the expected number of records) for new files. The expected number of records should be passed as INUMB. Alternatively, the size name, as describe in the DSSUTL open command documentation, can be passed in CSTR instead of INUMB. This call is ignored for existing files.
TABLE	This parameter sets the internal hash-address table type for new files. CSTR can be either 'DYNAMIC' or 'STABLE'. A stable table is primarily intended for databases that do not change in size frequently. A dynamic table is intended where the file size may vary considerably or where the ultimate file size is not known. A stable table reserves a large portion of space at the beginning of the file for the table (which is incrementally added in a dynamic table). This call is ignored for existing files.
COMP	When set to 'ON', prior to a call to ZCOFIL, this will cause regular-interval time series data to be "re-compressed" as it is copied. The compression method used will be that which matches pathname parts set in the file's compression header. If the file does not have a compression header, then all the data will be "un-compressed".
CAST	When set to 'ON', prior to a call to ZCAT, this will cause a status line to be written to MUNIT during cataloging. The status line is updated every ten records. MUNIT must be connected to a terminal or console (not a file) during the catalog.
SQST	When set to 'ON', prior to a call to ZCOFIL, this will cause a status line to be written to MUNIT during copying. The status line is updated during the copy. MUNIT must be connected to a terminal or console (not a file) during the copy.
MAP	When set to 'ON', a map output is written to unit MAPUNT when a new catalog is created. See the ZCAT subroutine documentation for more information. Before setting MAP to 'ON', call ZSET setting MAPUNT to a valid unit number. MAP is set to 'OFF' by default.
MAPUNIT	This sets the unit number for the MAP output when MAP is 'ON' and a new catalog is created. The unit must have been previously opened. See the ZCAT subroutine documentation for further information.

Example:

```
C     If this is a new file, set the hash size.
      CALL ZFNAME (CN, CNAME, NNAME, LEXIST)
C
      IF (.NOT.LEXIST) THEN
         CALL ZSET ('SIZE', ' ', 5000)
      ENDIF
C
      CALL ZOPEN (IFLTAB, CNAME, ISTAT)
      IF (ISTAT.NE.0) GO TO 900
C     Set the program name and output to 80 columns.
      CALL ZSET ('PROG', 'DATSTR', IDUM)
      CALL ZSET ('80COL', 'ON', IDUM)
C
C     . . .
C
C     Store regular-interval time series data.
C     Set the tag and the precision.
      MAXPRE = 0
      NVALS = 0
 100  CONTINUE
      READ (9,120,END=200) CLINE
 120  FORMAT (A)
C     Parse the line.
      CALL PARSLI (CLINE, 20, NFIELD, IBF, IEF, ILF)
C
      DO 140 I=1,NFIELD
         NVALS = NVALS + 1
         VALUES(NVALS) = XREAL (CLINE, IBF(I), ILF(I), IERR)
         IF (IERR.NE.0) GO TO 910
            N = INDEX (CLINE(IBF(I):IEF(I), '.')
            IF (N.GT.0) THEN
               J = ILF(I) - N
               IF (N.GT.MAXPRE) MAXPRE = N
         ENDIF
 140  CONTINUE
C
C     Go back and read the next value.
      GO TO 100
C
C     All the data has been read;  store it.
      CALL ZSET ('TAG', 'NF-FLOW', IDUM)
      CALL ZSET ('PREC', ' ', MAXPRE)
      IF (LDEBUG) CALL ZSET ('MLEVEL', ' ', 9)
      CALL ZSITS (IFLTAB, . . .
```

2.6 ZINQIR - Inquire About DSS Parameters

Purpose:

ZINQIR provides a means of determining what parameters or flags are set to. This includes items such as the message level, a record's last written date and time, and the number of records in the file.

Calling Sequence:

```
CALL ZINQIR (IFLTAB, CITEM, CSTR, INUMB)
```

Declarations:

```
INTEGER IFLTAB(600), INUMB
CHARACTER CITEM*4, CSTR*(*)
```

On MS-DOS microcomputers, INUMB must be INTEGER*4: INTEGER*4 INUMB

Argument Description:

IFLTAB	Input/ Output	The DSS workspace used to manage the DSS file. This is the same array used in the ZOPEN call.
CITEM	Input	The item to inquire about. Items may be abbreviated to four characters. A list of available items follows.
CSTR	Output	If the item inquired about is returned as a character string, it is returned in this variable.
INUMB	Output	If the item inquired about is returned as an integer number, it is returned in this variable.

Remarks:

ZINQIR will either return a character string, or an integer number; the other variable will be unchanged. Several items refer to the last record read (e.g., record version). If no records have been accessed, the variables will be undefined. The information returned is current to the time the file was last accessed by your program. If someone else is writing to the file at the same time, and you have not accessed the file for some time, some of the information returned may not be current (e.g., the number of records in the file).

Summary Table:

CITEM	Description	Variable
'80COL'	Returns 'ON' if the abbreviate messages to eighty column flag is set.	CSTR
'DEAD'	Returns the percentage of dead space in this file.	INUMB
'FDATE'	Returns the creation date of the DSS file.	CSTR
'FVERS'	Returns the DSS version of the file when it was created.	CSTR
'HSIZE'	Returns the hash table size.	INUMB
'MLEVEL'	Returns the message level.	INUMB
'MUNIT'	Returns the unit number used for message output.	INUMB
'NAME'	Returns the name of DSS file.	CSTR
'NREC'	Returns the number of records in the file.	INUMB
'PRECISION'	Returns the precision setting of the last record read.	INUMB
'PROGRAM'	Returns the name of the program for the last record read.	CSTR
'FLAGS'	Returns the data flag for the last record read.	INUMB
'RDATE'	Returns the last write date of the last record read.	CSTR
'READONLY'	Returns 'ON' if the file is in a read only mode.	CSTR
'RTIME'	Returns the last write time of the last record read.	CSTR
'RVERS'	Returns the version number of the last record read.	INUMB
'SIZE'	Returns the size of the DSS file in kilobytes.	INUMB
'TABLR'	Returns the hash table type of the file.	CSTR
'TAG'	Returns the tag of the last record read.	CSTR
'UNIT'	Returns the unit number of the DSS file.	INUMB
'VERSION'	Returns the DSS software version.	CSTR

Parameters

80COL	This returns 'ON' in CSTR if the abbreviate message output to eighty columns flag is set. If the flag is not set, 'OFF' is returned.
DEAD	This returns in INUMB the percentage of inactive space in the file, rounded to the nearest integer.
FDATE	This returns in CSTR the creation date of the DSS file, as recorded in the file header. The date is in a seven-character military style form (e.g., 20MAR84).
FVERS	This returns in CSTR the DSS version of the file when it was created, or last squeezed. The version is returned in a form similar to "6-FC".
HSIZE	This returns in INUMB the hash table size as a number of one through eight. The eight sizes correspond to the sizes discussed in the DSSUTL open command, where one represents 'tiny', and eight 'extra-huge'.

MLEVEL	This returns in INUMB the current message level. See the MLEVEL documentation in ZSET for information on the different levels.
MUNIT	This returns in INUMB the unit number for writing messages.
NAME	This returns in CSTR the name of the DSS file.
NREC	This returns in INUMB the number of records in the file, according to the last access (if someone else is writing to the file at the same time, the actual number may be different).
PRECISION	This returns in INUMB the precision setting of the last record read. If INUMB is zero, no precision has been set for that record. See ZSET for more information.
PROGRAM	This returns in CSTR the name of the program that stored data in the last record read.
FLAGS	This returns in INUMB the data flag of the last record read. INUMB will be one if data flags are used, otherwise it will be returned as zero.
RDATE	This returns in CSTR the last write date of the last record read. The date is returned in a military style (e.g., 20MAR78).
READONLY	This returns 'ON' in CSTR if the file mode is set to read only.
RTIME	This returns in CSTR the last write time of the last record accessed. The time is given in twenty-four hour clock time (e.g., 1630 for 4:30 p.m.).
RVERS	This returns in INUMB the version number of the last record accessed. The version number corresponds to the number of times that record has been written to.
SIZE	This returns in INUMB the size of the DSS file in kilobytes (according to the last access).
TABLE	This returns in CSTR 'DYNAMIC' if the hash table is dynamic, or 'STABLE' if the table type is stable.
TAG	This returns in CSTR the tag of the last record read.
UNIT	This returns in INUMB the unit number of the DSS file.
VERSION	This returns in CSTR the current DSS software version. Versions are in a form such as "6-EA".

2.7 ZFVER - Get a DSS File's Version

Purpose:

ZFVER determines the software version for an un-opened DSS file. This routine is used primarily in environments where older (incompatible) DSS version files may exist (for example both DSS Versions 5 and 6 exist on MS-DOS computers). The software version for an opened DSS file may be obtained by the subroutine ZINQIR.

Calling Sequence:

```
CALL ZFVER (CNAME, CVER, IVER)
```

Declarations:

```
CHARACTER CNAME*64, CVER*4
INTEGER IVER
```

Argument Description:

CNAME	Input	The name of the DSS file. The default file name extension will be used if no extension is passed (and the computer system uses extensions). The file must be un-opened.
CVER	Output	The four-character DSS version identifier (e.g., "6-FC"). If the file is not a DSS file, CVER will be returned blank filled.
IVER	Output	The DSS file version number, or status parameter if the file is not a DSS file. The possible values returned are:

IVER	**Description**
-3	The file is not a DSS file.
-2	Unable to access the file (but it exists).
-1	The file does not exist.
4	Version 4 file
5	Version 5 file
6	Version 6 file

Remarks:

ZFVER calls ZFNAME to determine if the file exists. If it does, it is temporarily opened, and key file identifiers are read and examined. ZFVER can also be used to determine if a file is a DSS file before calling ZOPEN.

Example:

```
C     Open a DSS file.  It is ok to open a new DSS file, but
C     don't open a non-DSS file, or an older version file.
C
      CHARACTER CNAME*64, CVER*4
C
      CALL ATTACH ( 0, 'DSS', ' ', 'NOP', CNAME, ISTAT)
      . . .
C
      CALL ZFVER (CNAME, CVER, IVER)
C
      IF (IVER.EQ.-3) THEN
         WRITE (6,20) CNAME
 20      FORMAT (' File ',A,' is not a DSS file!')
         GO TO 900
      ENDIF
C
      IF (IVER.EQ.-2) THEN
         WRITE (6,40) CNAME
 40      FORMAT (' Unable to Access File ',A)
         GO TO 900
      ENDIF
C
C     A IVER of -1 is ok (the file does not exist).
C
      IF (IVER.NE.6) THEN
         WRITE (6,60) CNAME, CVER
 60      FORMAT (' The file ',A,' is a version ',A,' file',/
     *   ' This program can only access DSS version 6 files.')
         GO TO 900
      ENDIF
C
      CALL ZOPEN (IFLTAB, CNAME, ISTAT)
```

3 Pathname Manipulation Subroutines

DSS records are referenced by their pathnames. A pathname consists of up to eighty characters and is, by convention, separated into six parts. The parts are referred to by the characters A, B, C, D, E, and F, and are delimited by a slash "/", as follows:

/A/B/C/D/E/F/

Each pathname part may contain up to thirty-two characters, with the total length of the pathname not exceeding eighty characters. Pathname parts may have embedded blanks (e.g., "RED RIVER" for the A part), but blanks prior to and following each part are removed (so that a blank will never be adjacent to a slash).

Valid pathname characters are the set of upper case characters, digits, the space character and the following characters: ! $ % & ( ) * + - . : ; < > ? [ ] { } \ | ~. The characters @ # = can also be used, but are discouraged because they conflict with other uses. Invalid pathname characters are the set of lower case characters, control characters (including the null character), the following characters: , ' " ` ^ and the "delete" character. Any lower case characters used in a pathname are translated to upper case by the DSS software. The forward slash (/) can only be used as a part separator.

Refer to the data conventions portion of the Overview section in the "HECDSS User's Guide and Utility Program Manuals" as to what each of the parts should contain.

Subroutine ZPATH constructs a pathname from the six-pathname parts. ZUPATH determines the beginning and ending position and length of each part of a pathname. ZUFPN (un-form pathname) returns each part of a pathname in a character variable. ZGPNP (get pathname parts) obtains pathname parts from a line where the parts are identified by the part letter and an equal sign (e.g., A=SCIOTO, B=SOUTH BEND). An example of using ZPATH, ZUFPN, and ZGPNP is provided in the ZGPNP documentation.

ZCHKPN examines a pathname to determine if it meets the requirements for a pathname. This includes determining if the pathname contains seven slashes, is equal to or less than eighty characters in length, and contains invalid characters (e.g., control codes).

3.1 ZPATH - Construct a Pathname

Purpose:

ZPATH constructs a DSS pathname from the six-pathname parts. ZPATH removes leading and trailing blanks from each part, and inserts a slash (/) between each part and at the beginning and end of the pathname.

Calling Sequence:

```
CALL ZPATH (CA, CB, CC, CD, CE, CF, CPATH, NPATH)
```

Declarations:

```
CHARACTER CA*32, CB*32, CC*32, CD*32, CE*32, CF*32, CPATH*80
INTEGER NPATH
```

Argument Description:

CA	Input	A character string containing the A part of the pathname. The part may be up to thirty-two characters in length. The part may have blanks before and after the part, which will be removed by ZPATH. Embedded blanks (e.g., "RED RIVER") are not removed. A null part should be specified by setting the part to all blanks (e.g., CA = ' ').
CB	Input	The B part of the pathname.
CC	Input	The C part of the pathname.
CD	Input	The D part of the pathname.
CE	Input	The E part of the pathname.
CF	Input	The F part of the pathname.
CPATH	Output	The completed pathname.
NPATH	Output	The number of characters in the pathname.

Remarks:

ZPATH replaces the functionality of subroutine ZFPN. Each pathname part may contain up to thirty-two characters, and the pathname may be up to eighty characters in length (including slashes). If the sum of the parts and slashes is greater than eighty characters, the last part(s) will be truncated so that the pathname is eighty characters.

An empty or null pathname part is specified by passing a blank string (' '). Parts should be blanked prior to calling ZPATH so that null characters (CHAR(0)) do not become accidentally imbedded in the pathname.

Example:

```
      CHARACTER CPATH*80, CA*32, CB*32, CF*32
C
      WRITE (6,*)'Enter Basin Name'
      READ (5,10,END=100,ERR=900) CA
 10   FORMAT (A)
      WRITE (6,*)'Enter Location Name'
      READ (5,10,END=100,ERR=900) CB
C
      CF = 'COMPUTED'
C
      CALL ZPATH (CA, CB, 'STAGE-DAMAGE', ' ', ' ', CF,
     * CPATH, NPATH)
C
      WRITE (6,20) CPATH(1:NPATH)
 20   FORMAT (' Pathname: ',A)
```

3.2 ZUPATH - Determine a Pathname's Part

Purpose:

ZUPATH determines the beginning and ending position, and length of each part of a pathname. This information is returned in three six-element integer arrays. The subroutine ZUFPN may be called instead of ZUPATH to return the pathname parts.

Calling Sequence:

```
CALL ZUPATH (CPATH, IBPART, IEPART, ILPART, ISTAT)
```

Declarations:

```
CHARACTER CPATH*80
INTEGER IBPART(6), IEPART(6), ILPART(6), ISTAT
```

Argument Description:

CPATH	Input	The pathname to process.
IBPART	Output	A six-element integer array returned with the beginning positions (in CPATH) of each of the pathname parts. IBPART(1) is the beginning position of the A part, IBPART(2) is the beginning position of the B part, etc.. The starting positions do not include slashes.
IEPART	Output	A six-element integer array returned with the ending position of each of the pathname parts. The ending position is the last character in each part, and does not include the slash.
ILPART	Output	A six-element integer array returned with the length of each of the pathname parts (excluding slashes). A null part is returned with a length of zero.
ISTAT	Output	A status parameter that is set to zero if there were no errors. If CPATH is not a valid pathname, ISTAT is returned as -1.

Remarks:

If a pathname part is null (ILPART() = 0), the beginning position and the ending position are both set to the position of the slash following the null part. A program should not attempt to use these positions when a part is null.

Example:

```
C     If a pathname meets the time series conventions,
C     print the D part and the time interval (in minutes).
      CHARACTER CPATH*80, CC
      INTEGER IBPART(6), IEPART(6), ILPART(6)
C
      READ (9,20,END=200,ERR=900) CPATH
 20   FORMAT (A)
C
      CALL ZUPATH (CPATH, IBPART, IEPART, ILPART, ISTAT)
C
      IF (ISTAT.NE.0) THEN
         WRITE (6,30) CPATH
 30      FORMAT (' Error:  Invalid Pathname Entered: ',A)
         GO TO 900
      ENDIF
C
C     Get the "C" part of the pathname and move into CC.
C     Be sure we do not have a null part.
      IF (ILPART(3).GT.0) THEN
         CC = CPATH(IBPART(3):IEPART(3))
      ELSE
C        This is a null part.  Blank fill.
         CC = ' '
         GO TO 200
      ENDIF
C
C     Get the time interval and date if it is time-series.
      IF ((ILPART(4).GT.0).AND.(ILPART(5).GT.0)) THEN
         JSTAT = 1
         CALL ZGINTL (INTL, CPATH(IBPART(5):IEPART(5)), NVALS, JSTAT)
         IF (JSTAT.GE.0) THEN
           WRITE (6,40) CC, CPATH(IBPART(4):IEPART(4)), INTL
 40        FORMAT (1X, A, ' data is available for ',A,/,
     *     ' with a time interval of ',I5,' minutes.')
         ENDIF
      ENDIF
```

3.3 ZUFPN - Spilt a Pathname into Separate Parts

Purpose:

ZUFPN takes a standard pathname and segments it into six parts. Each part is returned as a separate character variable.

Calling Sequence:

```
      CALL ZUFPN (CA, NA, CB, NB, CC, NC, CD, ND, CE, NE,
   *  CF, NF, CPATH, NPATH, ISTAT)
```

Declarations:

```
      CHARACTER CA*32, CB*32, CC*32, CD*32, CE*32, CF*32, CPATH*80
      INTEGER NA, NB, NC, ND, NE, NF, NPATH, ISTAT
```

Argument Description:

CA	Output	A character string containing the A part of the pathname. The part will be left justified, blank filled in CA. If the part is null, CA will be returned with all blanks.
NA	Output	The number of characters in the A part. NA will be set to zero if the part is null (CA is all blanks).
CB	Output	The B part of the pathname.
NB	Output	The number of characters in the B part.
CC	Output	The C part of the pathname.
NC	Output	The number of characters in the C part.
CD	Output	The D part of the pathname.
ND	Output	The number of characters in the D part.
CE	Output	The E part of the pathname.
NE	Output	The number of characters in the E part.
CF	Output	The F part of the pathname.
NF	Output	The number of characters in the F part.
CPATH	Input	The pathname, which is to be segmented.

NPATH	Input	The number of characters in the pathname.
ISTAT	Output	A status parameter that is set to zero if there were no errors. If CPATH is not a valid pathname, ISTAT is returned as -1.

Remarks:

If only one or two parts of the pathname are needed, use subroutine ZUPATH, which returns the part positions within the pathname. The pathname must follow the standard conventions (six parts, each part separated by a slash). Each part may contain up to thirty-two characters, and the pathname may contain up to eighty characters (including slashes). If a part is longer than the length of the corresponding character variable passed, the part will be truncated to fit the character variable.

Example:

```
C     Break apart a pathname and print its parts.
      CHARACTER CPATH*80, CA*32, CB*32, CC*32, CD*32, CE*32, CF*32
C
C     Get the pathname.
      READ (5,10) CPATH
 10   FORMAT (A)
C
C     Get the position of the last non-blank character.
      CALL CHRLNB (CPATH, NPATH)
      CALL ZUFPN (CA, NA, CB, NB, CC, NC, CD, ND, CE, NE,
     * CF, NF, CPATH, NPATH, ISTAT)
C
      IF (ISTAT.NE.0) THEN
         IF (NPATH.EQ.0) NPATH = 1
         WRITE (6,10) CPATH(1:NPATH)
 10      FORMAT (' Illegal Pathname: ',A)
         GO TO 900
      ENDIF
C
      WRITE (6,20) CA, CB, CC, CD, CE, CF
 20   FORMAT (' Part A: ',A,/,' Part B: ',A,/,
     *  ' Part C: ',A,/,' Part D: ',A,/,
     * ' Part E: ',A,/,' Part F: ',A)
C
```

3.4 ZGPNP - Get Pathname Parts

Purpose:

ZGPNP takes a character string (usually read from input) and searches for parts of a pathname. Pathname parts are identified by a part identifier (A, B, C, D, E, or F), followed by an equal sign, then the part. Each part must be delimited by either a comma and/or a blank. Imbedded blanks may be included in a part, but commas, equal signs, and invalid pathname characters cannot be. The line may contain extraneous information (e.g., a "ZR" identifier), as long as the parts are identifiable. An example line that might be processed by ZGPNP is:

```
ZR=IN  A=SCIOTO, B=SOUTH BEND  F=OBS,C=FLOW  D=
```

Calling Sequence:

```
CALL ZGPNP (CLINE, CA, CB, CC, CD, CE, CF, NPARTS)
```

Declarations:

```
CHARACTER CA*32, CB*32, CC*32, CD*32, CE*32, CF*32
CHARACTER CLINE*(*)
INTEGER NPARTS(6)
```

Argument Description:

CLINE	Input	The character string from which to extract the pathname parts. The string may contain information other than the pathname parts, as long as it can be distinguished. For example: ZR, XXX B=SOUTH FORK, XXX F=OBS (the parts are distinguishable) ZR B=SOUTH FORK XXX F=OBS (XXX cannot be distinguished from the "B" part).
CA	Output	The A part, if found. If the part is not found, CA is not altered (a previously set A part may be passed without alteration).
CB	Output	The B part, if found.
CC	Output	The C part, if found.
CD	Output	The D part, if found.
CE	Output	The E part, if found.
CF	Output	The F part, if found.

NPARTS	Input/ Output	A six-element array that returns the lengths of the parts found. The first element of NPARTS corresponds to the A part, the second element corresponds to the B part, etc. NPARTS may also be used so that specific parts will not be searched for. To not search for a part, set the corresponding NPARTS to -2. On return, if a part was not found, the corresponding NPARTS will be set to -1. If no parts were found, NPARTS(1) will be set to -10.

Remarks:

ZGPNP is meant to update parts of a pathname, and is often used along with ZUFPN and ZPATH. Unless a part is found, it will not be altered. ZGPNP searches for a part letter followed by an equal sign, so other information may be on the input line, as long as it is distinguishable. If a part is longer than the character variable passed, the part will be truncated to fit within the length of that variable.

Null parts may be specified by the part letter followed by an equal sign then a comma (if it is at the end of CLINE, the comma is not necessary). A null part variable is returned blank filled.

Example 1:

For the following code:

```
  NPARTS(4) = -2
  CALL ZGPNP (CLINE(1:80), CA, CB, CC, CD, CE, CF(1:10), NPARTS)
```

If CLINE contains:

```
  ZR=IN ZA=HI, B=SOUTH BEND, XX C=, D=01JAN1960 F=PLAN 2B-COMPUTED
```

ZGPNP returns:

```
  NPARTS(1) =  -1,  CA =
  NPARTS(2) =  10,  CB = SOUTH BEND
  NPARTS(3) =   0,  CC =
  NPARTS(4) =  -2,  CD =
  NPARTS(5) =  -1,  CE =
  NPARTS(6) =  10,  CF = PLAN 2B-CO
```

Note that ZA is not a valid part identifier, so no "A" part was returned. The B part is terminated by a comma, so the "XX" was ignored. The C part was set to 0, since a comma immediately followed the equal sign. No D part was returned because NPARTS(4) was preset to -2. The F part was truncated, because CF was limited to a length of ten characters (1:10).

Example 2:

The following example illustrates the use of subroutines ZPATH, ZUFPN, and ZGPNP. This code reads a line, determines if it is a pathname or pathname parts. Pathname parts are extracted via ZUFPN or ZGPNP; then a new pathname is constructed. This code may be used in a loop, allowing the user to specify a completely new pathname or just change certain parts.

```
      CHARACTER CLINE*80, CPATH*80
      CHARACTER CA*32, CB*32, CC*32, CD*32, CE*32, CF*32
      INTEGER NPARTS(6)
C
      DATA CA, CB, CC, CD, CE, CF /6*' '/
C
C     Get the pathname or parts.
      WRITE (6,*) 'Enter pathname, or pathname parts, or FINISH'
      READ (5,20) CLINE
 20   FORMAT (A)
C
C     Is this a FINISH command?
      IF (CLINE(1:3).EQ.'FIN') GO TO 800
C
C     Is this a pathname?
      IF (CLINE(1:1).EQ.'/') THEN
C
C        Yes it is.
         CALL ZUFPN (CA, NA, CB, NB, CC, NC, CD, ND, CE, NE,
     *   CF, NF, CLINE, 80, ISTAT)
         IF (ISTAT.NE.0) GO TO 910
C
      ELSE IF (INDEX(CLINE(1:20),'=')) THEN
C
C        The line appears to contain pathname parts.
         CALL ZGPNP (CLINE, CA, CB, CC, CD, CE, CF, NPARTS)
C
         IF (NPARTS(1).EQ.-10) THEN
           WRITE (6,*)'Invalid Pathname Line.'
           GO TO 800
         ENDIF
C
      ELSE
C
         WRITE (6,*)'Invalid Pathname Line.'
         GO TO 800
C
      ENDIF
C
```

```
C     Now that the pathname is in parts, (re)construct the pathname.
      CALL ZPATH (CA, CB, CC, CD, CE, CF, CPATH, NPATH)
C
      WRITE (6,40) CPATH(1:NPATH)
 40   FORMAT ( ' Pathname: ', A)
```

3.5 ZCHKPN - Check a Pathname

Purpose:

ZCHKPN examines a pathname to determine if it meets the requirements for a pathname. This includes determining if the pathname contains seven slashes, is equal or less than eighty characters in length, and contains any invalid characters (e.g., control codes). If any "null characters" are found, they are changed to blanks.

Calling Sequence:

```
CALL ZCHKPN (CPATH, NPATH, ISTAT)
```

Declarations:

```
CHARACTER CPATH*80
INTEGER NPATH, ISTAT
```

Argument Description:

CPATH	Input/ Output	The pathname to be checked.
NPATH	Input	The number of characters in CPATH
ISTAT	Output	A status parameter indicating the validity of the pathname. ISTAT is returned as zero if CPATH is a valid pathname. If ISTAT is less than zero, the pathname is invalid and should not be used. The possible status codes are:

ISTAT	Description
0	The pathname is valid
+6	One or more null characters were detected and converted to blank characters.
-1	The first character in the pathname is not a slash (/).
-2	The last character in the pathname is not a slash (/).
-3	The number of slashes within the pathname is not seven. (There must be exactly seven slashes.)
-4	There are fewer than seven characters in the pathname.
-5	There are more than eighty characters in the pathname.
-6	Illegal characters were found in the pathname. (They were not modified.)

Remarks:

ZCHKPN will print an error message to unit MUNIT if any errors are detected and the message level is three or greater. The pathname will be modified only if it contains null characters (which can usually be avoided by initializing all pathname parts to blanks in a data statement).

4 Time Series Subroutines

The following chapter describes the subroutines used to store and retrieve regular-interval and irregular-interval time series data.

Regular-interval time series data is data that occurs at a standard time interval. The date and time of each value is implied by its position within the data block. As described in "HECDSS User's Guide and Utility Program Manuals", Overview section, the A part of the pathname contains the group identifier, the B part provides the location, the C part carries the parameter, the D part is the block start date, the E part holds the time interval and the F part is an optional descriptor. The E part must be one of the following time intervals:

Valid Time Intervals (E Parts)	**Block Length**
1MIN, 2MIN, 3MIN, 4MIN, 5MIN 10MIN, 15MIN, 20MIN, 30MIN	One Day
1HOUR, 2HOUR, 3HOUR, 4HOUR, 6 HOUR, 8HOUR, 12HOUR	One Month
1DAY	One Year
1WEEK, 1MON	One Decade
1YEAR	One Century

Regular-interval time series data may be retrieved with either subroutine ZRRTS or ZRRTSX. ZRRTSX is the extended form of ZRRTS, and will obtain data flags, the user header array, and data compression information. Regular-interval time series data may be stored with either ZSRTS or ZSRTSX, where ZSRTSX is the extended version of ZSRTS. ZSRTSX can store the user header array and data flags, whereas ZSRTS cannot. Data compression parameters may also be specified with ZSRTSX. (Data compression cannot be applied when data flags are stored.) Missing values within a record are flagged by values of -901.0, and values for missing records are flagged with -902.0.

Irregular-interval time series data has an explicit date and time stored with each value. This data is retrieved with either subroutine ZRITS or ZRITSX. ZRITSX is the extended form of ZRITS, and will return data flags and the user header in addition to the data. Irregular-interval time series data may be stored in a DSS file with either ZSITS or ZSITSX. ZSITSX can store the user header array and data flags, whereas ZSITS cannot.

Most of the DSS time series routines use Julian dates, in days since 31DEC1899 (not days since the beginning of the year). This form of date provides an exact and relative easy means of dealing with time date information. For example, to increment the date by one day, one is added to the Julian date, whereas a more complex algorithm would be required for a military style date such as 28FEB1972. Julian dates can be negative, allowing for handling data in the 1800's. A Julian date can be converted to another style date (of which many forms are available) using the HECLIB subroutine JULDAT. Conversely, different styles of dates can be converted to Julian using the subroutine DATJUL. See the HECLIB documentation for more information on these subroutines.

Several subroutines pass time information in minutes past midnight. The time in minutes can be converted to a twenty-four hour military style time (e.g., 1430 is 2:30 p.m.) by the HECLIB subroutine M2IHM, and back to minutes with subroutine IHM2M. The time interval of the data is also given in minutes, regardless of the length of the interval.

A utility subroutine, ZGINTL, is used to convert a time interval (in minutes) to the E part of the pathname, and vice-versa. Subroutine ZOFSET will determine the time offset of regular-interval data. The time offset is defined as the length of time between the standard time for that interval and the actual time. For example, the time offset for daily data measured at 8:00 a.m. is 480 minutes (eight hours).

On MS DOS computers, the Julian dates, time interval, and time offset must always be declared as INTEGER*4.

4.1 ZRRTS - Retrieve Regular-Interval Time Series Data

Purpose:

ZRRTS is a short call to retrieve regular-interval time series data from a DSS file. The data retrieved may be based on a time window and can cross record boundaries (that is, it can read several records with different dates to retrieve the data specified), or it can read all the data in one record according to the pathname (with no time window). When reading data based on a time window, the D part is ignored, as ZRRTS forms pathnames with a D part determined by that time window. The time window is specified by variables CDATE, CTIME, and NVALS. If data flags, compression information, or the user header needs to be retrieved, use subroutine ZRRSTX, the extended version of this subroutine.

Calling Sequence:

```
     CALL ZRRTS (IFLTAB, CPATH, CDATE, CTIME, NVALS, VALUES,
   * CUNITS, CTYPE, IOFSET, ISTAT)
```

Declarations:

```
     INTEGER IFLTAB(600), NVALS, IOFSET, ISTAT
     REAL VALUES(NVALS)
     CHARACTER CPATH*80, CDATE*20, CTIME*4, CUNITS*8, CTYPE*8
```

On MS DOS microcomputers, the time offset must be INTEGER*4: INTEGER*4 IOFSET

Argument Description:

IFLTAB	Input/ Output	The DSS work space used to manage the DSS file. This is the same array used in the ZOPEN call.
CPATH	Input	The pathname of the data to read. The pathname must meet the regular-interval time series conventions (including a correct "E part"). With a time window specified, the "D part" (date part) will be ignored; as ZRRTS will form it internally (there may be several D parts, depending on the time window). If no time window is given, the D part must be provided. The length of CPATH is implicit (e.g., CPATH(1:NPATH)).
CDATE	Input	The beginning date of the time window. This can be in any one of the styles accepted by the HECLIB subroutine DATJUL (see the HECLIB documentation for the different date styles). If the data is to be retrieved based upon the date in the pathname (that is, no time window), CDATE should be blank (i.e., ' ').

CTIME	Input	The beginning time of the time window. This must be a standard twenty-four hour clock time (e.g., '1630'). If no time window is set (CDATE is blank), this argument is ignored.
NVALS	Input/ Output	The number of data values to retrieve. This defines the end of the time window. If no time window is given, NVALS must contain the dimension limit of array VALUES on input, and is returned with the number of data values actually read by ZRRTS (and must be a variable).
VALUES	Output	The data retrieved. This will be in a sequential order, with the first value having a date and time of CDATE and CTIME (unless no time window is given, whereas the first value will correspond to the date and time of the beginning of the record).
CUNITS	Output	The units of the data (e.g., 'FEET').
CTYPE	Output	The type of the data (e.g., 'PER-AVER').
IOFSET	Output	The time offset of the data in minutes. (If hourly data is recorded at fifteen minutes past the hour, the offset would be fifteen minutes.) If there is no offset, IOFSET will be returned as zero. Refer to the subroutine ZOFSET (at the end of this chapter) for more information about time offsets. The offset must be INTEGER*4 on MS DOS microcomputers.
ISTAT	Output	A status parameter indicating the success of the operation. If ISTAT is returned as zero, then the data was successfully read. If ISTAT is returned with a value between one and three, then data was retrieved, but some missing values were detected. If ISTAT is greater than ten, a fatal error occurred, and no data was returned. The possible values are:

ISTAT	**Description**
0	All data retrieved.
1	Some missing data was detected (-901.0).
2	Missing record(s) (-902.0), but some data was found.
3	Missing record(s) and missing data in the data set, however some data was found.
4	There was no data for this time window, but a record was read.
5	No records were found (the data is returned as all -902's).
>10	A "fatal" error occurred:
	11 The number of values requested was less than one.

12	A non-standard time interval was provided in the "E" part of the pathname.
15	The starting date or time was not recognized.
20	The data was not recognized as regular-interval time series.
24	The pathname given does not meet the regular-interval time series conventions.
53	The data could not be un-compressed.

Remarks:

CUNITS and CTYPE will contain the units and type for the last record read when reading several records. If no records were found (ISTAT=5), or a fatal error occurred, CUNITS and CTYPE will be unchanged.

If data flags or the user header needs to be retrieved, or compression information is required, use ZRRTSX, the extended version of this subroutine.

A debug trace will be printed when the message level (MLEVEL) is set to 8 via subroutine ZSET. This trace will print the pathname, dates, times, and other information used by the subroutine.

Example 1:

```
C     Retrieve 200 values starting from December 13, 1982,
C     then print them out.
      INTEGER IFLTAB(600), IBPART(6), IEPART(6), ILPART(6)
      INTEGER*4 JULS, JULE, INTL, IOFSET
      CHARACTER CPATH*80, CDATE*20, CTIME*4, CUNITS*8, CTYPE*8
      REAL VALUES(200)
C
C     Open the DSS file and get the pathname.
      CALL ZOPEN ( ...
C
C     Retrieve the data.
      NVALS = 200
      CALL ZRRTS (IFLTAB, CPATH, '13DEC82', '2400', NVALS,
     * VALUES, CUNITS, CTYPE, IOFSET, ISTAT)
      IF (ISTAT.GE.10) GO TO 900
      IF (ISTAT.GE.4)  GO TO 100
C
C     Get the time interval from the pathname.
      CALL ZUPATH (CPATH, IBPART, IEPART, ILPART, ISTAT)
      IF (ISTAT.NE.0) GO TO 910
      JSTAT = 1
      CALL ZGINTL (INTL, CPATH(IBPART(5):IEPART(5)), N, JSTAT)
      IF (JSTAT.NE.0) GO TO 920
C
C     Convert 13DEC82 to julian.
      CALL DATJUL ('13DEC82', JULS, IERR)
      ISTIME = 1440
C     Adjust for any time offset.
      CALL ZOFSET (JULS, ISTIME, INTL, 2, IOFSET)
C
C     Print the values, along with the date and time of each one.
      DO 80 I=1,200
         IDUM = INCTIM (INTL, 0, I-1, JULS, ISTIME, JULE, IETIME)
         CALL JULDAT (JULE, 0, CDATE, NDATE)
         IDUM = M2IHM (IETIME, CTIME)
         WRITE (6,40) CDATE(1:NDATE), CTIME, VALUES(I)
 40      FORMAT (1X,A,2X,A,F10.3)
 80   CONTINUE
```

Example 2:

```
C     Retrieve and print the last 60 values from the current time.
C
      INTEGER IFLTAB(600), IBPART(6), IEPART(6), ILPART(6)
      INTEGER*4 JULS, JULE, INTL, IOFSET
      CHARACTER CPATH*80, CDATE*20, CTIME*4, CUNITS*8, CTYPE*8
      REAL VALUES(60)
C
C
C     Open the DSS file and get the pathname.
      CALL ZOPEN ( ...
C
C     Get the time interval from the pathname.
      CALL ZUPATH (CPATH, IBPART, IEPART, ILPART, ISTAT)
      IF (ISTAT.NE.0) GO TO 910
      JSTAT = 1
      CALL ZGINTL (INTL, CPATH(IBPART(5):IEPART(5)), N, JSTAT)
      IF (JSTAT.NE.0) GO TO 920
C
C     Get the current date and time, in Julian days.
      CALL CURTIM ( JULE, IETIME)
C     Decrement it by 59 periods (60 values).
      IDUM = INCTIM ( INTL, 0, -59, JULE, IETIME, JULS, ISTIME)
C      Date style 104 is used here, but any style would be ok.
      CALL JULDAT (JULS, 104, CDATE, NDATE)
      IDUM = M2IHM (ISTIME, CTIME)
C
C     Now retrieve the data.
```

```
      NVALS = 60
      CALL ZRRTS (IFLTAB, CPATH, CDATE(1:NDATE), CTIME,
     * NVALS, VALUES, CUNITS, CTYPE, IOFSET, ISTAT)
      IF (ISTAT.GE.10) GO TO 900
      IF (ISTAT.GE.4)  GO TO 100
C
C     Adjust for any time offset.
      CALL ZOFSET (JULS, ISTIME, INTL, 2, IOFSET)
C
C     Print the values, along with the date and time of each one.
      DO 80 I=1,60
         IDUM = INCTIM (INTL, 0, I-1, JULS, ISTIME, JULE, IETIME)
         CALL JULDAT (JULE, 0, CDATE, NDATE)
         IDUM = M2IHM (IETIME, CTIME)
         WRITE (6,40) CDATE(1:NDATE), CTIME, VALUES(I)
 40      FORMAT (1X,A,2X,A,F10.3)
 80   CONTINUE
```

4.2 ZRRTSX - Retrieve Regular-Interval Time Series Data (Extended Version)

Purpose:

ZRRTSX is the extended call to retrieve regular-interval time series data from a DSS file. This subroutine will return data flags, the user header, and compression information (if available) along with data. If this additional information is not needed, use ZRRTS, the short form of this subroutine.

The data retrieved may be based on a time window and can cross record boundaries (that is, it can read several records with different dates to retrieve the data specified), or it can read all the data in one record according to the pathname (with no time window). When reading data based on a time window, the D part is ignored, as ZRRTSX forms pathnames with a D part determined by that time window. The time window is specified by variables CDATE, CTIME, and NVALS.

Calling Sequence:

```
      CALL ZRRTSX (IFLTAB, CPATH, CDATE, CTIME, NVALS, VALUES,
    * FLAGS, LFLAGS, LFREAD, CUNITS, CTYPE, HEADU, KHEADU, NHEADU,
    * IOFSET, ICOMP, ISTAT)
```

Declarations:

```
      INTEGER IFLTAB(600)
      INTEGER NVALS, KHEADU, NHEADU, ICOMP, IOFSET, ISTAT
      REAL VALUES(NVALS), HEADU(KHEADU), FLAGS(NVALS)
      LOGICAL LFLAGS, LFREAD
      CHARACTER CPATH*80, CDATE*20, CTIME*4, CUNITS*8, CTYPE*8
```

On MS DOS microcomputers, the time offset must be INTEGER*4: INTEGER*4 IOFSET

Argument Description:

IFLTAB	Input/ Output	The DSS work space used to manage the DSS file. This is the same array used in the ZOPEN call.
CPATH	Input	The pathname of the data to read. The pathname must meet the regular-interval time series conventions. With a time window specified, the D part (date part) will be ignored, as ZRRTSX will form it internally (there may be several D parts, depending on the time window). If no time window is given, the D part must be provided. The length of CPATH is implicit (e.g., CPATH(1:NPATH)).

CDATE	Input	The beginning date of the time window. This can be in any one of the styles accepted by the HECLIB subroutine DATJUL (see the HECLIB documentation for the different date styles). If the data is to be retrieved based upon the date in the pathname (that is, no time window), CDATE should be blank (i.e., ' ').
CTIME	Input	The beginning time of the time window. This must be a standard twenty-four hour clock time (e.g., '1630'). If no time window is set (CDATE is blank), this argument is ignored.
NVALS	Input/ Output	The number of data values to retrieve. This defines the end of the time window. If no time window is specified, NVALS must contain the dimension limit of array VALUES on input, and is returned with the number of data values actually read by ZRRTSX (and must be a variable).
VALUES	Output	The data retrieved. This will be in a sequential order, with the first value having a date and time of CDATE and CTIME (unless no time window is given, whereas the first value will correspond to the date and time of the beginning of the record).
FLAGS	Output	The thirty-two bit data flags retrieved. See the appendix to interpret the bit settings. If data flags are not to be retrieved, set LFLAGS to .FALSE. and FLAGS may be a dummy argument.
LFLAGS	Input	A logical flag indicating whether data flags should be retrieved, if available. Set this to .TRUE. if flags are to be retrieved, .FALSE. if flags are not to be retrieved.
LFREAD	Output	A logical flag indicating whether data flags were retrieved. If the data did not have flags, or LFLAGS was set to .FALSE., this variable will be returned as .FALSE. (and FLAGS will be unchanged). If flags were retrieved, this variable will be set to .TRUE.
CUNITS	Output	The units of the data (e.g., 'FEET').
CTYPE	Output	The type of the data (e.g., 'PER-AVER').
HEADU	Output	The optional user header array. This array usually may be decoded by subroutine ZUSTFH.
KHEADU	Input	The dimension of array HEADU. No more than KHEADU elements of the user header array will be retrieved. If you do not want to retrieve the user header, set this to zero.

NHEADU	Output	The number of elements in the user header actually retrieved. NHEADU will always be equal to or less than KHEADU.
IOFSET	Output	The time offset of the data in minutes. (If hourly data is recorded at fifteen minutes past the hour, the offset would be fifteen minutes.) If there is no offset, IOFSET will be returned as zero. Refer to the subroutine ZOFSET (at the end of this chapter) for more information about time offsets. The offset must be INTEGER*4 on MS DOS microcomputers.
ICOMP	Output	The data compression method used if this data was compressed. If the data was not compressed, ICOMP will be returned as zero. If ICOMP is greater than zero, more data compression information can be obtained by calling subroutine ZDCINF.
ISTAT	Output	A status parameter indicating the success of the operation. If ISTAT is returned with zero, then all the data was successfully read. If ISTAT is returned with a value between one and three, then data was retrieved, but some missing values were detected. If ISTAT is greater than ten, a fatal error occurred, and no data was returned. The possible values are:

ISTAT	**Description**	
0	All data retrieved.	
1	Some missing data was detected (-901.0).	
2	Missing record(s) (-902.0), but some data was found.	
3	Missing record(s) and missing data in the data set, however some data was found.	
4	There was no data for this time window, but a record was read.	
5	No records were found (no data is returned).	
>10	A "fatal" error occurred:	
	11	The number of values requested was less than one.
	12	A non-standard time interval was provided in the "E" part of the pathname.
	15	The starting date or time was not recognized.
	20	The data was not recognized as regular-interval time series.
	24	The pathname given does not meet the regular-interval time series conventions.
	53	The data could not be un-compressed.

Remarks:

CUNITS and CTYPE will contain the units and type for the last record read when reading several records. If no records were found (ISTAT=5), or a fatal error occurred, CUNITS and CTYPE will be unchanged.

If the data was stored with data flags, no data compression method will be returned, because data cannot be compressed when flags are used.

A debug trace will be printed when the message level (MLEVEL) is set to eight via subroutine ZSET. This trace will print the pathname, dates, times, and other information used by the subroutine.

Example:

(Note: This is a more "comprehensive" example of retrieving time series data. For a simpler example, see the ZRRTS documentation.)

```
C
C     Retrieve database on a user's time window, then print
C     the data and information about the data.
C
      PARAMETER (KHEADU=100, KDATA=1000)
      REAL VALUES(KDATA), HEADU(KHEADU), FLAGS(KDATA)
      INTEGER IFLTAB(600)
      INTEGER IBPART(6), IEPART(6), ILPART(6)
      INTEGER*4 JULS, JULE, INTL, IOFSET
      CHARACTER CLINE*80, CPATH*80, CDATE*20, CTIME*4
      CHARACTER CUNITS*8, CTYPE*8, CINFO*24, CLABEL*20, CITEM*20
      LOGICAL LFREAD, LBASEV
C
C     Open the DSS file.
      CALL ZOPEN (IFLTAB, ...
C
C     Get the pathname and its length.
      WRITE (6,*)'Enter the pathname'
      READ  (5,10) CPATH
 10   FORMAT (A)
C
C     Get the time window.
      WRITE (6,*)'Enter the time window'
      READ  (5,10) CLINE
      CALL GETIME (CLINE, 1, 80, JULS, ISTIME, JULE, IETIME, ISTAT)
      IF (ISTAT.NE.0) GO TO 900
C
C     Get the time interval from the pathname.
      CALL ZUPATH (CPATH, IBPART, IEPART, ILPART, ISTAT)
      IF (ISTAT.NE.0) GO TO 910
      JSTAT = 1
      CALL ZGINTL (INTL, CPATH(IBPART(5):IEPART(5)), N, JSTAT)
      IF (JSTAT.NE.0) GO TO 920
C
C     Compute the number of data values asked for.
      NVALS = NOPERS (INTL, 0, JULS, ISTIME, JULE, IETIME)
      IF (NVALS.LE.0) GO TO 930
      IF (NVALS.GT.KDATA) GO TO 940
C
C     Convert the date and time to character.
      CALL JULDAT (JULS, 104, CDATE, NDATE)
      I = M2IHM (ISTIME, CTIME)
C     Retrieve the data.
      CALL ZRRTSX (IFLTAB, CPATH, CDATE, CTIME, NVALS,
     * VALUES, FLAGS, .TRUE., LFREAD, CUNITS, CTYPE, HEADU,
     * KHEADU, NHEADU, IOFSET, ICOMP, ISTAT)
C
C     "Fatal" error?
      IF (ISTAT.GE.10) GO TO 950
C     No data?
      IF (ISTAT.GE.4) GO TO 960
C
C     Write pathname, units.
      CALL CHRLNB (CPATH, NPATH)
      WRITE (6,40) CPATH(1:NPATH), CUNITS, CTYPE
 40   FORMAT (' Pathname: ',A,/,' Units: ',A,T20,'Type: ',A)
C
C     Get more compression information (if used).
      IF (ICOMP.GT.0) THEN
         CALL ZDCINF (ICOMP, BASEV, LBASEV, ISIZE, IPREC, ISTAT)
         WRITE (6,60) ICOMP, LBASEV, BASEV, ISIZE, IPREC
 60      FORMAT (' Compression Method:',I3,'  User Base:',L2,
     *   ' Base: ',F6.1,/,' Size Allocated:',I2,'  Precision:',I3)
      ELSE
         WRITE (6,80)
 80      FORMAT (' No Compression Used.')
      ENDIF
C
C     Print the user header (if any).
      IF (NHEADU.GT.0) THEN
         NITEM = 0
         IPOS = 0
         WRITE (6,*)'Header:'
 100     CONTINUE
         CALL ZUSTFH (CLABEL, CITEM, NITEM, IPOS, HEADU, NHEADU,
     *   ISTAT)
```

```
            IF (ISTAT.NE.0) THEN
              WRITE (6,*)'Invalid User header.'
              GO TO 140
            ENDIF
            WRITE (6,120) CLABEL, CITEM
 120        FORMAT (1X,A,1X,A)
            IF (IPOS.GE.0) GO TO 100
         ENDIF
C
C        Adjust for any time offset.
 140     CONTINUE
            CALL ZOFSET (JULS, ISTIME, INTL, 2, IOFSET)
C        Print the values, along with the date and time of each one,
C        and any information from the header.
         DO 200 I=1,NVALS
            IDUM = INCTIM (INTL, 0, I-1, JULS, ISTIME, JULE, IETIME)
            CALL JULDAT (JULE, 0, CDATE, NDATE)
            IDUM = M2IHM (IETIME, CTIME)
C
C           Do we need to print data flag information?
            IF (LFREAD) THEN
            CALL GETBIT (FLAGS(I), 1, ISCRN)
            CALL GETBIT (FLAGS(I), 8, IMOD)
C
            IF (ISCRN.EQ.1) THEN
              IF (IMOD.EQ.1) THEN
                 CINFO = 'Screened and Modified'
              ELSE
                 CINFO = 'Screened'
              ENDIF
            ELSE
              CINFO = ' '
            ENDIF
C
         ELSE
            CINFO = ' '
         ENDIF
C
         WRITE (6,160) CDATE(1:NDATE), CTIME, VALUES(I), CINFO
 160     FORMAT (1X,A,' at ',A,';  ',F10.3,2X,A)
 200     CONTINUE
```

4.3 ZSRTS - Store Regular-Interval Time Series Data

Purpose:

ZSRTS is a short call to store regular-interval time series data in a DSS file. The data to be stored is based on a time window, which can cross record boundaries (that is, it can write several records with different D parts). Because the time window is specified by variables CDATE, CTIME, and NVALS, the D part of the pathname is ignored.

If data flags or a user header is to be stored along with the data, use ZSRTSX, the extended version of this subroutine. ZSRTSX also provides a means of specifying a data compression method and optional compression parameters. However, data may be compressed by ZSRTS if subroutine ZSCOMP is called just prior to ZSRTS, or if a file compression method is set and the required pathname parts match, or if the record already exists and is compressed.

Calling Sequence:

```
     CALL ZRRTS (IFLTAB, CPATH, CDATE, CTIME, NVALS, VALUES,
*    CUNITS, CTYPE, IOFSET, ISTAT)
```

Declarations:

```
     INTEGER IFLTAB(600), NVALS, IOFSET, ISTAT
     REAL VALUES(NVALS)
     CHARACTER CPATH*80, CDATE*20, CTIME*4, CUNITS*8, CTYPE*8
```

Argument Description:

IFLTAB	Input/ Output	The DSS work space used to manage the DSS file. This is the same array used in the ZOPEN call.
CPATH	Input	The pathname of the data to store. The pathname must meet the regular-interval time series conventions. The D part (date part) is ignored, as ZSRTS will form it internally from the time window. The length of CPATH is implicit (e.g., CPATH(1:NPATH)).
CDATE	Input	The beginning date of the time window (the date of the first value). This can be in any one of the styles accepted by the HECLIB subroutine DATJUL (see the HECLIB documentation for the different date styles).
CTIME	Input	The beginning time of the time window in twenty-four hour clock time (e.g., '1630'). Any time offset is implied by the date and time specified (for example, if daily data is measured at 8:00 a.m., then setting CTIME to '0800' implies an offset of eight hours or 480 minutes).

NVALS	Input/	The number of values to store. This defines the end of the time window.
VALUES	Input	The data to store. This must be in a sequential order, with the first value data having a date and time of CDATE and CTIME.
CUNITS	Input	The units of the data (e.g., 'FEET').
CTYPE	Input	The type of the data (e.g., 'PER-AVER').
IPLAN	Input	An argument to indicate whether to write over existing data or not. If IPLAN is set to zero, the data provided will always replace any existing data (with the same pathname at the same times).

IPLAN	**Description**
0	Always write over existing data.
1	Only replace missing data flags in the record (-901).
4	If an input value is missing (-901), do not allow it to replace a non-missing value.

ISTAT	Output	A status parameter indicating the success of the operation. If ISTAT is returned with zero, then all the data was successfully stored. The possible values are:

ISTAT	**Description**	
0	The data was successfully stored.	
4	All of the input data provided were missing data flags (-901).	
>10	A "fatal" error occurred:	
	11	The number of values to store (NVALS) is less than one.
	12	Unrecognized time interval (E part).
	15	The starting date or time is invalid.
	24	The pathname given does not meet the regular-interval time series conventions.
	51	Unrecognized data compression scheme (when ZSCOMP is called prior to ZSRTS). Valid schemes are zero to five.
	53	Invalid precision exponent specified for the delta data compression method (when ZSCOMP is called prior to ZSRTS). The precision exponent range is -6 to 6.

Remarks:

Missing data in the array VALUES should be flagged by setting those values to -901.0.

To set a data compression method to be used by ZSRTS, call subroutine ZSCOMP just prior to ZSRTS, or use subroutine ZSRTSX, in which compression information is passed is as arguments. A file compression method may also be set to compress data.

If data without data flags is merged with (or replaces) data with flags, then data flags (set to zero) will be added to the new data.

If the record exists and has a user header stored with it, ZSRTS will not change or delete that header.

A debug trace may be turned on by setting the message level (MLEVEL) to seven, eight, or nine via subroutine ZSET. Level 7 gives information regarding the arguments being passed. The higher levels provide information about the steps taking place inside ZSRTS.

Example:

```
C     A program has computed 86 time series data values.
C     Store them in a DSS file.
C
      INTEGER IFLTAB(600)
      CHARACTER CPATH*80, CDATE*20, CTIME*4
      REAL VALUES(86)
C
C     Open the DSS file and get the pathname.
      CALL ZOPEN ( ...
C
C     Assuming the time interval is in minutes,
C     convert it to a valid "E part".
      ISTAT = 2
      CALL ZGINTL (INTL, CE, IDUM, ISTAT)
      CALL ZPATH  ( ...
C
C     Convert the date from integer 12/24/83
C     to a character date.
      CALL YMDDAT ( IYR, IMON, IDAY, 0, CDATE, NDATE, IERR)
      IF (IERR.NE.0) GO TO 900
C     Convert the time from minutes to 24 hour clock time.
      IDUM = M2IHM (ITIME, CTIME)
C
C     Now store the data.
      CALL ZSRTS (IFLTAB, CPATH, CDATE, CTIME, 86, VALUES,
     * 'FEET', 'PER-AVER', 0, ISTAT)
      IF (ISTAT.GT.0)  GO TO 900
```

4.4 ZSRTSX - Store Regular-Interval Time Series Data (Extended Version)

Purpose:

ZSRTSX is the extended call to store regular-interval time series data in a DSS file. ZSRTSX will store data flags and a user header along with the data. In addition, a data compression method and related parameters may be set with ZSRTSX. If data flags or a user header is not to be stored or data compression is not used, call ZSRTS, the short form of this subroutine.

Data is stored based on a time window, which can cross record boundaries (that is, it can write several records with different D parts). Because the time window is specified by variables CDATE, CTIME, and NVALS, the D part of the pathname is ignored.

Calling Sequence:

```
     CALL ZSRTSX (IFLTAB, CPATH, CDATE, CTIME, NVALS, VALUES,
 *   FLAGS, LFLAGS, CUNITS, CTYPE, HEADU, NHEADU, IPLAN,
 *   ICOMP, BASEV, LBASEV, LHIGH, IPREC, ISTAT)
```

Declarations:

```
     INTEGER IFLTAB(600)
     INTEGER NVALS, NHEADU, ICOMP, IPREC, IPLAN, ISTAT
     REAL VALUES(NVALS), FLAGS(NVALS), HEADU(NHEADU), BASEV
     CHARACTER CPATH*80, CDATE*20, CTIME*4, CUNITS*8, CTYPE*8
     LOGICAL LFLAGS, LBASEV, LHIGH
```

Argument Description:

IFLTAB	Input/ Output	The DSS work space used to manage the DSS file. This is the same array used in the ZOPEN call.
CPATH	Input	The pathname of the data to store. The pathname must meet the regular-interval time series conventions. The D part (date part) is ignored, as ZSRTSX will form it internally from the time window. The length of CPATH is implicit (e.g., CPATH(1:NPATH)).
CDATE	Input	The beginning date of the time window (the date of the first value). This can be in any one of the styles accepted by the HECLIB subroutine DATJUL (see the HECLIB documentation for the different date styles).
CTIME	Input	The beginning time of the time window. This must be a standard twenty-four hour clock time (e.g., '1630'). Any time

		offset is implied by the date and time specified (for example, if daily data is measured at 8:00 a.m., then setting CTIME to '0800' implies an offset of eight hours or 480 minutes).
NVALS	Input/	The number of values to store. This defines the end of the time window.
VALUES	Input	The data to store. This must be in a sequential order, with the first value having a date and time of CDATE and CTIME.
FLAGS	Input	An array containing thirty-two bit data flags. If flags are not to be stored then set LFLAGS to .FALSE. and FLAGS may be a dummy argument.
LFLAGS	Input	A logical flag indicating whether data flags are to be stored or not. To store the FLAGS array, set LFLAGS to .TRUE. If data flags are not to be stored, set this to .FALSE.
CUNITS	Input	The units of the data (e.g., 'FEET').
CTYPE	Input	The type of the data (e.g., 'PER-AVER').
HEADU	Input	The optional user header array. Information should be placed in this array by subroutine ZSTFH. If no user header is to be stored, this may be a dummy argument and NHEADU should be set to zero.
NHEADU	Input	The number of elements in the user header array HEADU. If no user header information is to be stored, set this to zero. To have ZSRTSX not change an existing record's user header, set this to -1. For new records, a -1 (as well as zero) will store no user header.
IPLAN	Input	An argument to indicate whether to write over existing data or not. If IPLAN is set to zero, the data provided will always replace any existing data (with the same pathname at the same times).

IPLAN	**Description**
0	Always write over existing data.
1	Only replace missing data flags in the record (-901).
4	If an input value is missing (-901), do not allow it to replace a non-missing value.

ICOMP	Input	The data compression method to use. To use the default file method, set this to zero. To disallow compression for this data, set ICOMP to -1. The compression methods are described in Chapter 10. If data flags are stored the data will not be compressed.

BASEV	Input	When the delta data compression method is used, the base value may be specified by setting this argument to the base value and LBASEV to .TRUE. If the delta method is not used, this argument is ignored.
LBASEV	Input	A logical flag indicating if the argument BASEV has been set. To let the compression software select a base value, set this argument to .FALSE.
LHIGH	Input	When the delta data compression method is used, setting LHIGH to .TRUE. will pre-allocate two bytes of storage per data value. If LHIGH is set to .FALSE., the compression software will select the storage size based on the data. If the delta method is not used, this argument is ignored.
IPREC	Input	When the delta data compression method is used; this defines the precision exponent of the data (required). The precision exponent may range from -6 to +6. If the delta method is not used, this argument is ignored.
ISTAT	Output	A status parameter indicating the success of the operation. If ISTAT is returned with zero, then all the data was successfully stored. The possible values are:

ISTAT	**Description**	
0	The data was successfully stored.	
4	All of the input data provided were missing data flags (-901).	
>10	A "fatal" error occurred:	
	11	The number of values to store (NVALS) is less than one.
	12	Unrecognized time interval (E part).
	15	The starting date or time is invalid.
	24	The pathname given does not meet the regular-interval time series conventions.
	51	Unrecognized data compression scheme. Valid schemes are zero to five.
	53	Invalid precision exponent specified for the delta data compression method. The precision exponent range is -6 to 6.

Remarks:

Missing data in the array VALUES should be flagged by setting those values to -901.0.

If data with data flags is merged with (or replaces) data without data flags, then data flags (set to zero) will be added to the old data before the merge. If data without data flags is merged with (or replaces) data with flags, then data flags (set to zero) will be added to the new data. If data flags are used, data compression is disabled.

If ICOMP is set to -1, the data will not be compressed no matter what the default file data compression settings are. If ICOMP is -1, and the record already exists and is compressed, the entire record will be un-compressed (and stored un-compressed).

A debug trace may be turned on by setting the message level (MLEVEL) to seven, eight, or nine via subroutine ZSET. Level 7 gives information regarding the arguments being passed. The higher levels provide information about the steps taking place inside ZSRTSX.

Example:

```
C     Store hourly elevation values that are provided
C     on a daily report (24 values per report).
C     Assume the daily report has been opened as unit 9,
C     and the report appears somewhat like the following:
C
C     SOUTH BASIN, MARTIN LAKE REPORT, 5/20/90
C     1123.44 1123.48 1124.21 1124.56 1124.99
C     1125.08 1125.12 1125.18 ...
C     END
C
C
      INTEGER IFLTAB(600)
      INTEGER IBF(2), IEF(20), ILF(20)
      CHARACTER CPATH*80, CDATE*12
      CHARACTER CBASIN*32, CLOC*32, CLINE*80
      LOGICAL LBASEV, LHIGH
      REAL VALUES(100), BASEV, HEADU, FLAGS
C
C     Open the dss file.
      CALL ZOPEN (IFLTAB, ...
C
C     Read the basin, location name, and the date of the first
C     value from the report.
      READ (9, 20, ERR=900, END=990) CBASIN, CLOC, CDATE
C
C     Construct the pathname (no date part is needed).
      CALL ZPATH (CBASIN, CLOC, 'ELEV', ' ', '1HOUR', 'OBS',
     * CPATH, NPATH)
C
C     Read the data.
      NVALS = 0
 30   CONTINUE
      READ (9, 40, ERR=910, END=800) CLINE
 40   FORMAT (A)
```

```
C     Have we reached the end of the data?
      IF (INDEX(CLINE,'END').GT.0) GO TO 100
C     Parse the line.
      CALL PARSLI (CLINE, 20, NFIELD, IBF, IEF, ILF)
      DO 50 I=1,NFIELD
         NVALS = NVALS + 1
C        Convert the character number to a real number.
         VALUES(NVALS) = XREAL(CLINE, IBF(I), ILF(I), IERR)
         IF (IERR.NE.0) VALUES(NVALS) = -901.0
 50   CONTINUE
      GO TO 30

 100  CONTINUE
C     This data is to be compressed.
C     Elevation data is normally compressed using the delta method.
C     Because this data is updated daily, we should set a base
C     value which would be a reasonable minimum elevation that might
C     be recorded (say middle of conservation).  This value is
C     needed only for the first time the month/block is written to.
      ICOMP = 2           !  (Delta compression method)
      BASEV  = 1023.0     !  (base value, probably obtained from
      LBASEV = .TRUE.     !  an external table)
      LHIGH = .TRUE.      !  (allocate 2 bytes of space for elevation)
      IPREC = -2          !  (store to the nearest hundredth of a foot)
C
C     Now store the data.
      CALL ZSRTSX (IFLTAB, CPATH, CDATE, '0100', NVALS,
     * VALUES, FLAGS, .FALSE., 'FEET', 'INST-VAL', HEADU, 0,
     * 0, ICOMP, BASEV, LBASEV, LHIGH, IPREC, ISTAT)
C
       IF (ISTAT.GT.0) GO TO 940
```

4.5 ZRITS - Retrieve Irregular-Interval Time Series Data

Purpose:

ZRITS is a short call to retrieve irregular-interval time series data from a DSS file. The data retrieved may be based on a time window and can cross record boundaries (that is, it can read several records with different dates to retrieve the data specified), or it can read all the data in one record according to the pathname (with no time window). When reading data based on a time window, the D part is ignored, as ZRITS forms pathnames with a D part determined by the time window.

If data flags or the user header needs to be retrieved, use subroutine ZRITSX, the extended version of this subroutine. ZRITSX also has the capability to retrieve the value previous to the time window and the value subsequent to the time window.

Calling Sequence:

```
      CALL ZRITS (IFLTAB, CPATH, JULS, ISTIME, JULE, IETIME,
     *  ITIMES, VALUES, KVALS, NVALS, JBDATE, CUNITS, CTYPE, ISTAT)
```

Declarations:

```
      INTEGER IFLTAB(600), KVALS, NVALS, ISTAT
      INTEGER JULS, ISTIME, JULE, IETIME, JBDATE, ITIMES(KVALS)
      REAL VALUES(KVALS)
      CHARACTER CPATH*80, CUNITS*8, CTYPE*8
```

On MS DOS microcomputers, the time offset must be INTEGER*4:

```
      INTEGER*4 JULS, JULE, JBDATE, ITIMES(KVALS)
```

On HARISS computers, the time offset must be INTEGER*6:

```
      INTEGER*6 ITIMES(KVALS)
```

Argument Description:

IFLTAB	Input/ Output	The DSS workspace used to manage the DSS file. This is the same array used in the ZOPEN call.
CPATH	Input	The pathname of the data to read. The pathname must meet the irregular-interval time series conventions (including a correct E part). With a time window specified, the D part (date part) will be ignored, as ZRITS will form it internally (there may be several D parts, depending on the time window). If no time window is given, the D part must be provided. The length of CPATH is implicit (e.g., CPATH(1:NPATH)).

JULS	Input	The Julian date of the start of the time window. This is days since December 31, 1899, not since the beginning of the current year. If no time window is specified, this argument is ignored (see ISTIME).
ISTIME	Input	The starting time of the time window, in minutes past midnight (for midnight ISTIME would be 1440, not zero). To read the entire record (with no time window set), set ISTIME to -2. The D part of the pathname will be used to define the time window.
JULE	Input/	The Julian date of the end of the time window in days since December 31, 1899. If no time window is set, this argument is ignored.
IETIME	Input	The ending time of the time window in minutes past midnight. If no time window is set, this argument is ignored.
ITIMES	Output	An array containing the relative date/times of the data values, in a one-to-one correspondence. The times are given in minutes since the base date (JBDATE), and can be converted into Julian dates and times using the procedure described in the remarks section.
VALUES	Output	The values retrieved. The date/time of each value is provided in array ITIMES. Both arrays VALUES and ITIMES must be dimensioned to KVALS.
KVALS	Input	The dimension of arrays VALUES and ITIMES, or (if desired) the maximum number of data values to retrieve. No more than KVALS values will be retrieved. If the message level (MLEVEL set via ZSET) is five or greater, a warning message will be printed when the KVALS limit has been reached.
NVALS	Output	The number of values retrieved. Arrays ITIMES and VALUES will contain NVALS elements.
JBDATE	Output	The Julian base date (in days since Dec. 31, 1899), usually equivalent to the D part of the first pathname. This date, in conjunction with the ITIMES array, gives the date/time of each data value.
CUNITS	Output	The units of the data (e.g., 'FEET').
CTYPE	Output	The type of the data (e.g., 'PER-AVER').
ISTAT	Output	A status parameter indicating the success of the operation. If ISTAT is returned with zero, then the data was successfully

read. If ISTAT is greater than ten, a fatal error occurred. The possible values are:

ISTAT	**Description**
0	The data was successfully stored.
1	The number of data values requested (according to the time window) exceeds KVALS. The ITIMES and VALUES arrays will contain KVALS values.
4	No data found (pathname not found). The output arguments are undefined.
>10	A "fatal" error occurred:
20	The data was not recognized as irregular-interval time series.
21	An internal buffer array is not large enough to read the record. (This will seldom occur as the same array is used to store the data, and the error would be detected at that time.)
24	The pathname does not meet the irregular-interval time series conventions.

Remarks:

The base date combined with the ITIMES array provide the date and time of each data value. The ITIMES array is returned with minutes from JBDATE for each value. This can be converted to a Julian date and time with the subroutine DATCLL. An example use of DATCLL is:

```
      INTEGER*4 JUL(KVALS), ITIMES(KVALS), JBDATE
      INTEGER MINS(KVALS)
      . . .
      CALL ZRITS ( . . .
C
      DO 20 I=1,NVALS
         CALL DATCLL (JBDATE, ITIMES(I), JUL(I), MINS(I))
 20   CONTINUE
      . . .
```

Earlier versions of DSS stored fractions of a day instead of minutes for the time array. This caused precision difficulties on thirty-two bit machines. A minimum of a thirty-two bit word size for the ITIMES array will allow a relative time range of up to 4085 years (2^{31} minutes).

CUNITS and CTYPE will contain the units and type for the last record read (when reading several records). If no data was found (ISTAT=4), or a fatal error occurred, CUNITS and CTYPE will be unchanged.

If data flags or the user header needs to be retrieved, or the previous or next data value (relative to the time window) is needed, use ZRITSX, the extended version of this subroutine.

A debug trace may be turned on by setting the message level (MLEVEL) to seven, eight, or nine via subroutine ZSET. Level 7 gives information regarding the arguments being passed, whereas the higher levels provide information about the steps taking place inside ZRITS.

Example:

```
C     Retrieve the past 60 days of data values and print their dates,
C     times and values.
C
      PARAMETER (KVALS=1000)
      REAL VALUES(KVALS)
      INTEGER*4 ITIMES(KVALS), JULS, JULE, JUL, JBDATE
      INTEGER IFLTAB(600)
      CHARACTER CPATH*80, CUNITS*8, CTYPE*8, CDATE*20, CTIME*4
C
C     Open the DSS file and get the pathname.
      CALL ZOPEN (IFLTAB, . . .
C
C     Get the current julian date and time.
      CALL CURTIM (JULE, IETIME)
C     Decrement it by 60 days.
      IDUM = INCTIM (1440, 0, -60, JULE, IETIME, JULS, ISTIME)
      ISTIME = 1440
C
C     Retrieve the data.
      CALL ZRITS (IFLTAB, CPATH, JULS, ISTIME, JULE,
     * IETIME, ITIMES, VALUES, KVALS, NVALS, JBDATE, CUNITS,
     * CTYPE, ISTAT)
C
C     Check for errors.
      IF (ISTAT.GE.10) GO TO 900
      IF (ISTAT.EQ.4) GO TO 100
C
C     Print out the data.
      CALL CHRLNB (CPATH, NPATH)
      WRITE (6,20) CPATH(1:NPATH), CUNITS, CTYPE
 20   FORMAT (. . .
      DO 60 I=1, NVALS
C        Convert the times array into a regular date and time.
         CALL DATCLL (JBDATE, ITIMES(I), JUL, IMIN)
         CALL JULDAT (JUL, 0, CDATE, NDATE)
         IDUM = M2IHM (IMIN, CTIME)
         WRITE (6,40) CDATE(1:NDATE), CTIME, VALUES(I)
 40      FORMAT (1X,A,',  ',A,';  ',F8.2)
 60   CONTINUE
```

4.6 ZRITSX - Retrieve Irregular-Interval Time Series Data (Extended Version

Purpose:

ZRITSX is the extended call to retrieve irregular-interval time series data from a DSS file. This subroutine will return data flags and the user header along with the data. In addition, ZRITSX will, if desired, return the value preceding and/or following the time window. If this additional information is not needed, use ZRITS, the short form of this subroutine.

The data retrieved by ZRITSX may be based on a time window and can cross record boundaries (that is, it can read several records with different dates), or it can read all the data in one record with no time window. When reading data based on a time window, the "D part" is ignored, as ZRITSX forms pathnames with a D part determined by the time window.

Calling Sequence:

```
      CALL ZRITSX (IFLTAB, CPATH, JULS, ISTIME, JULE, IETIME,
     *   ITIMES, VALUES, KVALS, NVALS, JBDATE, FLAGS, LFLAGS, LFREAD,
     *   CUNITS, CTYPE, HEADU, KHEADU, NHEADU, INFLAG, ISTAT)
```

Declarations:

```
      INTEGER IFLTAB(600), KVALS, NVALS, ISTAT
      INTEGER JULS, ISTIME, JULE, IETIME, JBDATE
      INTEGER ITIMES(KVALS), KHEADU, NHEADU, INFLAG
      REAL VALUES(KVALS), FLAGS(KVALS), HEADU(KHEADU)
      CHARACTER CPATH*80, CUNITS*8, CTYPE*8
      LOGICAL LFLAGS, LFREAD
```

On MS-DOS microcomputers, the Julian dates and the time array must be INTEGER*4:

```
      INTEGER*4 JULS, JULE, JBDATE, ITIMES(KVALS)
```

On HARRIS computers, the time array must be INTEGER*6:

```
      INTEGER*6 ITIMES(KVALS)
```

Argument Description:

IFLTAB	Input/ Output	The DSS workspace used to manage the DSS file. This is the same array used in the ZOPEN call.
CPATH	Input	The pathname of the data to read. The pathname must meet the irregular-interval time series conventions (including a correct "E part"). With a time window specified, the D part (date part) will be ignored, as ZRITSX will form it internally. If no time

		window is given, the D part must be provided. The length of CPATH is implicit (e.g., CPATH(1:NPATH)).
JULS	Input	The Julian date of the start of the time window. This is days since December 31, 1899 (not since the beginning of the current year). If no time window is specified, this argument is ignored (see ISTIME).
ISTIME	Input	The starting time of the time window, in minutes past midnight (for midnight ISTIME would be 1440, not zero). To use no time window and read the entire record, set ISTIME to -2. The D part of the pathname will be used to define the time window.
JULE	Input	The Julian date of the end of the time window in days since December 31, 1899. If no time window is set, this argument is ignored.
IETIME	Input	The ending time of the time window in minutes past midnight. If no time window is set, this argument is ignored.
ITIMES	Output	An array containing the relative date/times of the data values, in a one-to-one correspondence. The times are given in minutes since the base date (JBDATE), and can be converted into Julian dates and minutes since midnight as discussed in the remarks section.
VALUES	Output	The values retrieved. The date/time of each value is provided in array ITIMES. Both arrays VALUES and ITIMES must be dimensioned to KVALS.
KVALS	Input	The dimension of arrays VALUES and ITIMES, or (if desired) the maximum number of data values to retrieve. No more than KVALS values will be retrieved. If the message level is five (MLEVEL) or greater, a warning message will be printed when the KVALS limit has been reached.
NVALS	Output	The number of values retrieved. Arrays ITIMES and VALUES will contain NVALS elements.
JBDATE	Output	The Julian base date (in days since Dec. 31, 1899), usually equivalent to the D part of the pathname. This date, in conjunction with the ITIMES array, gives the date/time of each data value.
FLAGS	Output	The thirty-two bit data flags retrieved. This must be dimensioned to KVALS. See the appendix to interpret the bit

		settings. If data flags are not to be retrieved, set LFLAGS to .FALSE. and FLAGS may be a dummy argument.
LFLAGS	Input	A logical flag indicating whether data flags should be retrieved, if available. Set this to .TRUE. if flags are to be retrieved, .FALSE. if flags are not to be retrieved.
LFREAD	Output	A logical flag indicating whether data flags were retrieved. If the data did not have flags, or LFLAGS was set to .FALSE., this variable will be returned as .FALSE. (and FLAGS will be unchanged). If flags were retrieved, this variable will be set to .TRUE.
CUNITS	Output	The units of the data (e.g., 'FEET').
CTYPE	Output	The type of the data (e.g., 'PER-AVER').
HEADU	Output	The optional user header array. This array usually may be decoded by subroutine ZUSTFH.
KHEADU	Input	The dimension of array HEADU. No more than KHEADU elements of the user header array will be retrieved. If you do not want to retrieve the user header, set KHEADU to zero.
NHEADU	Output	The number of elements in the user header actually retrieved. NHEADU will always be equal to or less than KHEADU.
INFLAG	Input	A flag indicating if the value preceding and/or following the time window should be retrieved. This is valid only when a time window is provided (i.e., ISTIME is not set to -2). Valid INFLAG values are:

INFLAG	**Description**
0	Normal - retrieve data based on the time window.
1	Retrieve the value (and its date/time) preceding the time window in addition to the data within the time window.
2	Retrieve the value (and its date/time) following the time window in addition to the data within the time window.
3	Retrieve both values (and their date/time) preceding and following the time window in addition to the data within the time window.

ISTAT	Output	A status parameter indicating the success of the operation. If ISTAT is returned with zero, then the data was successfully read. If ISTAT is greater than ten, a fatal error occurred. The possible values are:

ISTAT	Description
0	The data was successfully stored.
1	The number of values requested (according to the time window) exceeds KVALS. The ITIMES and VALUES arrays will contain KVALS values.
4	No data found (pathname not found). The output arguments are undefined.
>10	A "fatal" error occurred:
20	The data was not recognized as irregular-interval time series.
21	An internal buffer array is not large enough to read the record. (This will seldom occur as the same array is used to store the data, and the error would be detected at that time.)
24	The pathname does not meet the irregular-interval time series conventions.

Remarks:

The base date combined with the ITIMES array provide the date and time of each data value. The ITIMES array is returned with minutes from JBDATE for each value. This can be converted to a Julian date and time with the subroutine DATCLL. An example use of DATCLL is:

```
      INTEGER*4 JUL(KVALS), ITIMES(KVALS), JBDATE
      INTEGER MINS(KVALS)
      . . .
      CALL ZRITSX ( . . .
C
      DO 20 I=1,NVALS
         CALL DATCLL (JBDATE, ITIMES(I), JUL(I), MINS(I))
 20   CONTINUE
      . . .
```

Earlier versions of DSS stored fractions of a day instead of minutes for the time array. This caused precision difficulties on thirty-two bit machines. A minimum of a thirty-two bit word size for the ITIMES array will allow a relative time range of up to 4085 years (2^{31} minutes).

When a preceding or subsequent value to the time window is requested via INFLAG, ZRITSX will search up to one record preceding or following the records within the data block. If no value is found, that point is not returned (i.e., a missing data flag is not returned).

CUNITS and CTYPE will contain the units and type for the last record read (when reading several records). If no data was found (ISTAT=4), or a fatal error occurred, CUNITS and CTYPE will be unchanged.

A debug trace may be turned on by setting the message level (MLEVEL) to seven, eight, or nine via subroutine ZSET. Level 7 gives information regarding the arguments being passed, whereas the higher levels provide information about the steps taking place inside ZRITSX.

Example:

```
C     Retrieve cumulative precipitation data from January 5, 1990
C     until January 25, 1990, then convert and print incremental
C     precipitation for those times.
C
      PARAMETER (KVALS=1000)
      REAL VALUES(KVALS), FLAGS(KVALS)
      INTEGER*4 ITIMES(KVALS), JULS, JULE, JUL, JBDATE
      INTEGER IFLTAB(600)
      CHARACTER CPATH*80, CUNITS*8, CTYPE*8
      CHARACTER CDATE1*20, CTIME1*4, CDATE2*20, CTIME2*4
      LOGICAL LFREAD
C
C     Open the DSS file and get the pathname.
      CALL ZOPEN (IFLTAB, . . .
C

C     Convert Jan 5 '90 and Jan 25 '90 to julian.
      CALL DATJUL ('JAN 5, 1990', JULS, IERR)
      IF (IERR.NE.0) GO TO 900
      CALL DATJUL ('JAN 25, 1990', JULE, IERR)
      IF (IERR.NE.0) GO TO 900
      ISTIME = 0001
      IETIME = 1400
C
C     Retrieve the previous and following data values.
      INFLAG = 3
C
C     Retrieve the data.
      CALL ZRITSX (IFLTAB, CPATH, JULS, ISTIME, JULE,
     * IETIME, ITIMES, VALUES, KVALS, NVALS, JBDATE, FLAGS, .TRUE.,
     * LFREAD, CUNITS, CTYPE, IDUM, 0, NDUM, INFLAG, ISTAT)
C
C     Check for errors.
      IF (ISTAT.GE.10) GO TO 900
      IF (ISTAT.EQ.4) GO TO 100
C
C      Convert and print the data.
      CALL CHRLNB (CPATH, NPATH)
      WRITE (6,20) CPATH(1:NPATH), CUNITS, CTYPE
 20   FORMAT (. . .
C
```

```
      DO 60 I=1, NVALS-1
C       Check for invalid precipitation data.
        IF ((VALUES(I).LT.0).OR.(VALUES(I+1).LT.0)) GO TO 60
C
C       Compute incremental from cumulative.
        PREINC = VALUES(I+1) - VALUES(I)
C
C       Convert the times of both into regular dates and times.
        CALL DATCLL (JBDATE, ITIMES(I), JUL, IMIN)
        CALL JULDAT (JUL, 0, CDATE1, NDATE1)
        IDUM = M2IHM (IMIN, CTIME1)
        CALL DATCLL (JBDATE, ITIMES(I+1), JUL, IMIN)
        CALL JULDAT (JUL, 0, CDATE2, NDATE2)
        IDUM = M2IHM (IMIN, CTIME2)
C
        WRITE (6,40) CDATE1(1:NDATE1), CTIME1, CDATE2(1:NDATE2),
     *  CTIME2, PREINC
 40     FORMAT (' From ',A,' at ',A,' through ',A,' at ',A,'  ;',F6.2)
C
         IF (LFREAD) THEN
           CALL GETBIT (FLAGS(I), 8, IEST)
           CALL GETBIT (FLAGS(I+1), 8, JEST)
           IF ((IEST.GT.0).OR.(JEST.GT.0)) WRITE (6,*) '(Estimated)'
        ENDIF
 60   CONTINUE
```

4.7 ZSITS - Store Irregular-Interval Time Series Data

Purpose:

ZSITS is a short call to store irregular-interval time series data in a DSS file. The data to be stored is based on an implied time window which can cross record boundaries (that is, ZSITS can write several records with different D parts). The time window is implied by the date and time of the first and last data values (which is to be provided in the ITIMES array).

Irregular-interval time series data is stored with times to the nearest minute. Data for times of less than a minute cannot be stored with this convention. The times of the data must be in ascending order, and no value may have the same exact time as another (you cannot have two data points for the same time in a record).

If data flags or a user header is to be stored with the data, use ZSITSX, the extended version of this subroutine.

Calling Sequence:

```
      CALL ZSITS (IFLTAB, CPATH, ITIMES, VALUES, NVALS, JBDATE,
   *  CUNITS, CTYPE, INFLAG, ISTAT)
```

Declarations:

```
      INTEGER IFLTAB(600), ITIMES(NVALS), NVALS, JBDATE, INFLAG, ISTAT
      REAL VALUES(NVALS)
      CHARACTER CPATH*80, CUNITS*8, CTYPE*8
```

On MS DOS microcomputers, the base date and the time array must be INTEGER*4:

```
      INTEGER*4 JBDATE, ITIMES(KVALS)
```

On HARRIS computers, the time array must be INTEGER*6:

```
      INTEGER*6 ITIMES(KVALS)
```

Argument Description:

IFLTAB	Input/ Output	The DSS workspace used to manage the DSS file. This is the same array used in the ZOPEN call.
CPATH	Input	The pathname of the data to store. The pathname must meet the irregular-interval time series conventions. The D part (date part) is ignored, as ZSITS will form it internally. The length of CPATH is implicit (e.g., CPATH(1:NPATH)).
ITIMES	Input	The array containing the relative date/times of the data values, in a one-to-one correspondence to the data. These values are to be

		in minutes from the base date (JBDATE), and can be generated from standard dates and times by the methods discussed in remarks (following). The times must be in ascending order, and no two times can be the same.
VALUES	Input	The values to store. The date/time of each value must be defined in array ITIMES.
NVALS	Input	The number of values to store. Arrays ITIMES and VALUES must contain NVALS elements.
JBDATE	Input	The Julian base date (in days since Dec. 31, 1899), which when combined with each element of ITIMES will give the Julian date and time for that value. All numbers in ITIMES must be relative to this value.
CUNITS	Input	The units of the data (e.g., 'FEET').
CTYPE	Output	The type of the data (e.g., 'PER-AVER').

INFLAG	Input	INFLAG is a flag to indicate whether the data should be replaced or merged with existing data. Replace will replace all the data between the implied time window (time of first and last data). Merge will combine the data with the data already stored. (Merging data replaces data occurring at the same time and inserts data at new times.)
		INFLAG = 0 to merge data.
		INFLAG = 1 to replace data.

ISTAT	Output	A status parameter indicating the success of the operation. If ISTAT is returned with zero, then all the data was successfully stored. If ISTAT is greater than ten, a fatal error occurred. The possible values are:

ISTAT		**Description**
0		The data was successfully stored.
4		No data was given to store (NVALS was zero).
>10		A "fatal" error occurred:
	21	An internal buffer array is not large enough to store this number of data values. If this error occurs, the time-block identified by in the "E part" of the pathname spans too long of a time, and holds more data values than the internal buffers can accommodate. The time-block should be changed to the next lower size (e.g., from "IR-MONTH" to "IR-DAY").
	24	The pathname does not meet the irregular-interval time series conventions.

ISTAT	Description
30	The times of the data values are not in an ascending order, or two values occur at the same time.

Remarks:

Generally the base date (JBDATE) is the Julian date of the first value (although it does not have to be as long as ITIMES will produce the correct date). The ITIMES array can be computed from the Julian date (JUL) and time (IMIN) of the data values by the following procedure (where JBDATE has been determined earlier):

```
INTEGER*4 JBDATE, JUL, ITIMES(NVALS)
INTEGER IMIN

ITIMES(I) = ((JUL - JBDATE) * 1440) + IMIN
```

Note that this math must usually be done in large integers (INTEGER*4 or INTEGER*6).

Earlier versions of DSS stored fractions of a day instead of minutes for the time array. This caused precision difficulties on thirty-two bit machines. A minimum of a thirty-two bit word size for the ITIMES array will allow a relative time range of up to 4085 years (2^{31} minutes).

With reference to INFLAG and data already present in the DSS file, replace will replace all the data within the implied time window, while merge will combine the two data sets, only replacing those values that occur at exactly the same time (within one minute of significance). INFLAG has no meaning for a new record. Usually the replace mode is used for editing data, and the merge mode is used for adding new data to the record.

If you are updating data where the first or last value might be deleted (for example an editing process where the first data might be removed), an explicit time window may be specified by setting the beginning time of the time window in the first element of ITIMES, and setting the corresponding data value to -902.0. The end of the time window is indicated in a similar manner, by setting the ending time in the last element in ITIMES and the corresponding value to -902.0. ZSITS will ignore the -902.0, but use the times to delete any data between that time and the next (or preceding) value in the array. (This cannot be used to delete an entire record, just data within a record.)

If data without data flags is merged with (or replaces) data with flags, then data flags (set to zero) will be added to the new data.

A debug trace may be turned on by setting the message level (MLEVEL) to nine via subroutine ZSET.

Example:

```
C     Read Irregular-interval time series data from an ASCII file, then
C     store it in DSS.  The input file might appear like the following:
C     3/12/90, 0800,  32.25
C     3/13/90, 1200,  33.14
C     END
C     Set a data precision (used by DSSUTL) by counting the number of
C     digits to the right of the decimal point.
C
      PARAMETER (KVALS=1000)
      INTEGER*4 ITIMES(KVALS), JBDATE, JUL
      INTEGER IFLTAB(600), IBF(20), IEF(20), ILF(20)
      REAL VALUES(KVALS)
      CHARACTER CLINE*80, CPATH*80, CUNITS*8, CTYPE*8
C
C     Open DSS file and get the pathname.
      CALL ZOPEN (IFLTAB, ...
C
 20   CONTINUE
      READ (9,40,END=200) CPATH, CUNITS, CTYPE
 40   FORMAT ( ...
C
      NVALS = 0
      MAXPRE = 0
 60   CONTINUE
C     Read a line from the input.
      READ (9,80,END=200) CLINE
 80   FORMAT (A)
C     Did we reach the end of the data yet?
      IF (INDEX(CLINE,'END').GT.0) GO TO 100
C
C     Parse the line.
      CALL PARSLI (CLINE, 20, NFIELD, IBF, IEF, ILF)
      IF (NFIELD.NE.3) GO TO 900
C
C     Get the date and time from this line.
      CALL DATJUL (CLINE(IBF(1):IEF(1)), JUL, IERR)
      IF (IERR.NE.0) GO TO 900
      IMIN = IHM2M (CLINE(IBF(2):IEF(2)))
      IF (IMIN.LT.0) GO TO 900
C     Compute the relative time of this value.
      NVALS = NVALS + 1
      IF (NVALS.EQ.1) JBDATE = JUL
      ITIMES(NVALS) = ((JUL - JBDATE) * 1440) + IMIN
C
C     Get the data value.
```

```
      VALUES(NVALS) = XREAL (CLINE, IBF(3), ILF(3), IERR)
      IF (IERR.NE.0) GO TO 900
C
C     Determine the data precision.
      N = INDEX (CLINE(IBF(3):IEF(3)), '.')
      IF (N.GT.0) THEN
         J = ILF(3) - N
         IF (J.GT.MAXPRE) MAXPRE = J
      ENDIF
C
C     Go back and read the next value.
      GO TO 60
C
C     All the data has been read;  store it.
 100  CONTINUE
      IF (NVALS.LE.0) GO TO 800
C
C     Set the data precision.
      CALL ZSET ('PREC', ' ', MAXPRE)
C
      CALL ZSITS (IFLTAB, CPATH, ITIMES, VALUES, NVALS,
    * JBDATE, CUNITS, CTYPE, 0, ISTAT)
      IF (ISTAT.NE.0) GO TO 900
      GO TO 20
```

4.8 ZSITSX - Store Irregular-Interval Time Series Data (Extended Version)

Purpose:

ZSITSX is the extended call to store irregular-interval time series data in a DSS file. This subroutine will store data flags and a user header along with the data. If flags or the user header will not be stored, use ZSITS, the short form of this subroutine.

The data stored by ZSITSX is based on implied time window which can cross record boundaries (that is ZSITSX can write several records with different D parts). The time window is implied by the date and time of the first and last data values (which is to be provided in the ITIMES array).

Irregular-interval time series data is stored with times to the nearest minute. Data for times of less than a minute cannot be stored with this convention. The times of the data must be in an ascending order, and no value may have the same exact time as another (you cannot have two data points for the same time in a record).

Calling Sequence:

```
      CALL ZSITSX (IFLTAB, CPATH, ITIMES, VALUES, NVALS, JBDATE,
   *  FLAGS, LFLAGS, CUNITS, CTYPE, HEADU, NHEADU, INFLAG, ISTAT)
```

Declarations:

```
      INTEGER IFLTAB(600), ITIMES(NVALS), NVALS, JBDATE
      INTEGER NHEADU, INFLAG, ISTAT
      REAL VALUES(NVALS), FLAGS(NVALS), HEADU(NHEADU)
      CHARACTER CPATH*80, CUNITS*8, CTYPE*8
      LOGICAL LFLAGS
```

On MS DOS microcomputers, the base date and the time array must be INTEGER*4:

```
INTEGER*4 JBDATE, ITIMES(KVALS)
```

On HARRIS computers, the time array must be INTEGER*6:

```
INTEGER*6 ITIMES(KVALS)
```

Argument Description:

IFLTAB	Input/ Output	The DSS workspace used to manage the DSS file. This is the same array used in the ZOPEN call.
CPATH	Input	The pathname of the data to store. The pathname must meet the irregular-interval time series conventions. The D part (date

		part) is ignored, as ZSITSX will form it internally. The length of CPATH is implicit (e.g., CPATH(1:NPATH)).
ITIMES	Input	The array containing the relative date/times of the data values, in a one-to-one correspondence to the data. These values are to be in minutes from the base date (JBDATE), and can be generated from standard dates and times by the methods discussed in remarks (following). The times must be in ascending order, and no two times can be the same.
VALUES	Input	The values to store. The date/time of each value must be defined in array ITIMES.
NVALS	Input	The number of values to store. Arrays ITIMES and VALUES must contain NVALS elements.
JBDATE	Input	The Julian base date (in days since Dec. 31, 1899), which when combined with each element of ITIMES will give that data values total Julian date and time. All values in ITIMES must be relative to this value.
FLAGS	Input	An array containing thirty-two bit data flags. If flags are not to be stored then set LFLAGS to .FALSE. and FLAGS may be a dummy argument.
LFLAGS	Input	A logical flag indicating whether data flags are to be stored or not. To store the FLAGS array, set LFLAGS to .TRUE. If flags are not to be stored, set this to .FALSE.
CUNITS	Input	The units of the data (e.g., 'FEET').
CTYPE	Input	The type of the data (e.g., 'PER-AVER').
HEADU	Input	The optional user header array. Information should be placed in this array by subroutine ZSTFH. If no user header is to be stored, this may be a dummy argument and NHEADU should be set to zero.
NHEADU	Input	The number of elements in the user header array HEADU. If no header information is to be stored, set this to zero. If the record exists and is stored with a user header, that header may be retained by setting this to the negative of the dimension of HEADU. Up to the absolute value of that dimension number of header elements will be retained (and also retrieved in HEADU). If the record does not exist, then no header will be stored when NHEAU is less than zero.

INFLAG	Input	INFLAG is a flag to indicate whether the data should be replaced or merged with existing data. Replace will replace all the data between the implied time window (time of first and last data). Merge will combine the data with the data already stored. (Merging data replaces data occurring at the same time, while inserting data that are for new times.)
		INFLAG = 0 to merge data.
		INFLAG = 1 to replace data.

ISTAT	Output	A status parameter indicating the success of the operation. If ISTAT is returned with zero, then all the data was successfully stored. If ISTAT is greater than ten, a fatal error occurred. The possible values are:

ISTAT	**Description**
0	The data was successfully stored.
4	No data was given to store (NVALS was zero).
>10	A "fatal" error occurred:
21	An internal buffer array is not large enough to store this number of data values. If this error occurs, the time-block identified by in the "E part" of the pathname spans too long of a time, and holds more data values than the internal buffers can accommodate. The time-block should be changed to the next lower size (e.g., from "IR-MONTH" to "IR-DAY").
24	The pathname does not meet the irregular-interval time series conventions.
30	The times of the data values are not in an ascending order, or two values occur at the same time.

Remarks:

Generally the base date (JBDATE) is the Julian date of the first value (although it does not have to be as long as ITIMES will produce the correct date). The ITIMES array can be computed from the Julian date (JUL) and time (IMIN) of the data values by the following procedure (where JBDATE has been determined earlier):

```
INTEGER*4 JBDATE, JUL, ITIMES(NVALS)
INTEGER IMIN

ITIMES(I) = ((JUL - JBDATE) * 1440) + IMIN
```

Note that this math must usually be done in large integers (INTEGER*4 or INTEGER*6).

Earlier versions of DSS stored fractions of a day instead of minutes for the time array. This caused precision difficulties on thirty-two bit machines. A minimum of a thirty-two bit word size for the ITIMES array will allow a relative time range of up to 4085 years (2^{31} minutes).

With reference to INFLAG and data already present in the DSS file, replace will replace all the data within the implied time window, while merge will combine the two data sets, only replacing those values that occur at exactly the same time (within one minute of significance). INFLAG has no meaning for a new record. Usually the replace mode is used for editing data, and the merge mode is used for adding new data to the record.

If you are updating data where the first or last value might be deleted (for example an editing process where the first data might be removed), an explicit time window may be specified by setting the beginning time of the time window in the first element of ITIMES, and setting the corresponding data value to -902.0. The end of the time window is indicated in a similar manner, by setting the ending time in the last element in ITIMES and the corresponding value to -902.0. ZSITSX will ignore the -902.0, but use the times to delete any data between that time and the next (or preceding) value in the array. (This cannot be used to delete an entire record, just data within a record.)

If data with data flags is merged with (or replaces) data without data flags, then data flags (set to zero) will be added to the old data before the merge. If data without data flags is merged with (or replaces) data with flags, then data flags (set to zero) will be added to the new data.

A debug trace may be turned on by setting the message level (MLEVEL) to nine via subroutine ZSET.

Example:

```
C     Do a range check on data stored in a DSS file.
C     Read the pathnames, and the time window from an external file.
C
      PARAMETER (KVALS=4000, KHEADU=500)
      INTEGER*4 ITIMES(KVALS), JULS, JULE, JBDATE
      INTEGER IFLTAB(600)
      REAL VALUES(KVALS), HEADU(KHEADU), FLAGS(KVALS)
      CHARACTER CLINE*80, CPATH*80, CUNITS*8, CTYPE*8
      LOGICAL LFREAD
C
C     Open the DSS file.
      CALL ZOPEN (IFLTAB, ...
C
C     Read the time window.
      READ (5, 20) CLINE
 20   FORMAT (A)
      CALL GETIME (CLINE, 1, 80, JULS, ISTIME, JULE,  IETIME, IST)
      IF (IST.NE.0) GO TO 900
C
```

```
C      Read irregular-interval time series pathnames.
 100   CONTINUE
       READ (5, 20, END=800) CPATH
C
C      Retrieve the data.
       CALL ZRITSX (IFLTAB, CPATH, JULS, ISTIME, JULE, IETIME, ITIMES,
     * VALUES, KVALS, NVALS, JBDATE, FLAGS, .TRUE., LFREAD, CUNITS,
     * CTYPE, HEADU, KHEADU, NHEADU, 0, ISTAT)
       IF (ISTAT.NE.0) GO TO 100


C      Now do a range check.
C      If the value is less than zero, reject it.
C      If it is greater than 30, mark it questionable, greater than 50
C      reject it.  (Bit 2 is ok, 4 is questionable, bit 5 is rejected.)
C
       DO 120 I=1,NVALS
C        If we did not read flags, zero out the flag.
C        (Note that on some computers, a zero may set some bits on,
C        and FLAGS must be zeroed by equivalencing to an integer.)
         IF (.NOT.LFREAD) FLAGS(I) = 0.0
C        Does it fail to be a positive value?
         IF (VALUES(I).LT.0.0) THEN
            CALL SETBIT (FLAGS(I), 5)
C        Is it greater than 30?
         ELSE IF (VALUES(I).GT.30.0) THEN
C           Is it greater than 50?
            IF (VALUES(I).GT.50.0) THEN
              CALL SETBIT (FLAGS(I), 5)
            ELSE
C             Data is questionable.
              CALL SETBIT (FLAGS(I), 4)
            ENDIF
         ELSE
C           Passes the range check, mark ok.
            CALL SETBIT (FLAGS(I), 2)
         ENDIF
 120   CONTINUE
C
C      Test complete.  Re-store the data.
       CALL ZSITSX (IFLTAB, CPATH, ITIMES, VALUES, NVALS, JBDATE,
     * FLAGS, TRUE., CUNITS, CTYPE, HEADU, NHEADU, 0, ISTAT)
       IF (ISTAT.NE.0) GO TO 900
C
C      Get the next set of data.
       GO TO 100
```

4.9 ZGINTL - Get Time Series Interval

Purpose:

Given the E part from a regular-interval time series pathname, ZGINTL will compute the time interval in minutes and the maximum number of data values contained in a block. Conversely, given a time interval in minutes, ZGINTL will generate a standard E part. If the E part meets the irregular-interval time series conventions, a status parameter will be returned indicating so (although no time interval will be returned).

Calling Sequence:

```
CALL ZGINTL (INTL, CE, NVALS, ISTAT)
```

Declarations:

```
INTEGER INTL, NVALS, ISTAT
CHARACTER CE*32
```

MS DOS microcomputers, the time interval INTL must be declared as INTEGER*4.

```
INTEGER*4 INTL
```

Argument Description:

INTL	Input/ Output	The time interval in minutes. When generating the E part, this will be input. When computing the interval from the E part, this will be output.
CE	Input/ Output	The E part. When generating the E part, this will be output. When computing the interval from the E part, this will be input.
NVALS	Output	The number of data values in a regular-interval time series block for this interval.
ISTAT	Input/ Output	ISTAT is used both as a flag for input, and a status parameter for output. For input, ISTAT indicates whether to convert INTL to an E part, or convert the E part to INTL.

As Input:

ISTAT	**Description**
1	Get the integer interval (INTL) from the character E part (CE).
2	Get the character E part (CE) from the integer interval (INTL) in minutes.

As Output:

ISTAT	**Description**
0	E part or INTL meets the regular-interval time series conventions.
1	E part meets the irregular-interval time series conventions (no interval returned).
-1	The E part is not recognized as time series.

Remarks:

ZGINTL can be used to help determine the type of data from the pathname. Because ISTAT is used both as an input and an output parameter, make sure that it is a variable that is set properly before calling ZGINTL.

Example:

```
C      Get the time interval from a pathname.
C
       CALL ZUFPN (..., CE, NE, ...
                 or, alternatively
       CALL ZGPNP (..., CE, ...
C
C      Get the time interval in minutes from the E part.
       ISTAT = 1
       CALL ZGINTL (INTL, CE, ND, ISTAT)
C
C      If ISTAT is returned as -1, the pathname is not time series.
       IF (ISTAT.EQ.-1) GO TO 900
C      Is the record irregular-interval time series?
       IF (ISTAT.EQ.1) GO TO 100
C      Regular-interval time series data.
```

4.10 ZOFSET - Determine the Time Offset of Time Series Data

Purpose:

ZOFSET will compute the interval offset from the standard interval time for regular-interval time series data. For example, the standard times for daily data are at 2400 hours (midnight), but data that is recorded at 0800 has an offset of 480 minutes (eight hours). ZOFSET can also change the Julian date and time to the standard time for that interval. Another capability of ZOFSET is to adjust the date and time according to the offset given.

Calling Sequence:

```
CALL ZOFSET (JUL, ITIME, INTL, IFLAG, IOFSET)
```

Declarations:

```
INTEGER JUL, ITIME, INTL, IFLAG, IOFSET
```

On MS DOS microcomputers, the Julian date, time interval and time offset must be INTEGER*4:

```
INTEGER*4 JUL, INTL, IOFSET
```

Argument Description:

JUL	Input/ Output	The Julian date, in days from 31DEC1899.
ITIME	Input/ Output	The time, in minutes past midnight.
INTL	Input	The time interval, in minutes. This must correspond to the standard regular-interval time series conventions.
IFLAG	Input	A flag indicating whether the date and time should be changed to a standard time:

IFLAG	Description
0	Only compute the offset (don't change JUL or ITIME).
1	Compute the offset and change the date and time to the standard date and time for that interval.
2	Adjust JUL and ITIME according to the offset provided (JUL and ITIME will be changed to a standard date and time first, then adjusted).

IOFSET	Input/ Output	The computed time offset, in minutes. The offset is the number of minutes between the standard time and the given time (JUL,

ITIME). If IFLAG is 2, then this argument must contain the offset to adjust JUL and ITIME with.

Remarks:

If the date/time is changed to a standard date/time, it is adjusted up. For example, a daily data value must occur between 0001 hours, and 2400 hours (1440 minutes). If ZOFSET changes the date and time to standard, it will become 2400 hours (1440 minutes).

Example 1 :

```
C      Check for a time offset of a date and time.
       CALL ZOFSET (JUL, ITIME, INTL, 0, IOFSET)
C
C      Now call an interpolation routine to compute data
C      for the standard time interval.
       IF (IOFSET.GT.0) THEN
         RATIO = REAL(IOFSET)/REAL(INTL)
         CALL INTERP (...
```

In the above example, if:

```
INTL   = 1440  (1 day)
JUL    = 29348 (May 8, 1980)
ITIME  = 420   (7 a.m.)
```

then IOFSET would be returned:

```
IOFSET  = 420
```

if IFLAG were set to 1 (adjust date/time), then the following variables would be returned:

```
JUL    = 29348 (May 8, 1980)
ITIME  = 2400   (midnight.)
INTL   = 420
```

Example 2 :

```
C      Adjust the date and time for data returned by ZRRTS
C      according to the offset, so that values are printed
C      with the correct time.
C
       CALL ZRRTS (..., IOFSET, ...)
C
C      Print the values and their date and time.
       DO 80 I=1,NVALS
         IDUM = INCTIM (INTL, 0, I-1, JULS, ISTIME, JULE, IETIME)
```

```
C        Adjust the date and time according to the offset.
         CALL ZOFSET ( JULE, IETIME, INTL, 2, IOFSET)
         CALL JULDAT (JULE, 0, CDATE, NDATE)
         IDUM = M2IHM (IETIME, CTIME)
         WRITE (6,40) CDATE(1:NDATE), CTIME, VALUES(I)
 40      FORMAT (1X,A,2X,A,F10.3)
 80    CONTINUE
```

5 Paired Data Subroutines

Paired data is a group of data that represents a two variable relationship. Typical examples are data that make up a curve (e.g., a rating table or a flow-frequency curve). Several sets of data may be stored in the same record if one of the variables is the same. For example, several elevation-damage curves may be stored in the same record, where the curves may be residential, commercial, etc. However, a stage-damage curve and a stage-flow curve should not be stored in the same record. A scale associated with the data set may be one of three types: linear, logarithmic, or probability.

Paired data is exchanged between a program and DSS in a singly dimensioned array. This usually requires that data be transferred from a doubly dimensioned array (with X and Y ordinates) into a singly dimensioned array. Examples of such a procedure are provided.

A label may be given to each data set. Labels are typically used to differentiate between curves within the same record. For example, an elevation-damage record may contain three curves whose labels might be "RESIDENTIAL", "AGRICULTURAL", and "COMMERCIAL".

Paired data is retrieved by subroutine ZRPD and stored with subroutine ZSPD. Prior to February 1987, paired data was stored and retrieved with subroutines ZWRITE and ZREAD, respectively. Because DSS version 6 uses an internal header to store information about the data, ZWRITE and ZREAD can no longer be used. The older subroutines ZGTPFD and ZPTPFD may still be used, although they maybe somewhat more cumbersome than ZRPD and ZSPD.

Further paired data conventions may be found in the Overview section of the "HECDSS User's Guide and Utility Program Manuals".

5.1 ZRPD - Retrieve Paired Data

Purpose:

ZRPD retrieves paired (curve) data from a DSS file. The curve's labels and the user header may be retrieved in addition to the data.

Calling Sequence:

```
      CALL ZRPD (IFLTAB, CPATH, NORD, NCURVE, IHORIZ,
     * C1UNIT, C1TYPE, C2UNIT, C2TYPE, VALUES, KVALS, NVALS,
     * CLABEL, KLABEL, LABEL, HEADU, KHEADU, NHEADU, ISTAT)
```

Declarations:

```
      INTEGER IFLTAB(600), NORD, NCURVE, IHORIZ, KVALS, NVALS
      INTEGER KLABEL, KHEADU, NHEADU, ISTAT
      REAL VALUES(KVALS), HEADU(KHEADU)
      CHARACTER CPATH*80, CLABEL(KLABEL)*12
      CHARACTER C1UNIT*8, C1TYPE*8, C2UNIT*8, C2TYPE*8
      LOGICAL LABEL
```

Argument Description:

IFLTAB	Input/ Output	The DSS work space used to manage the DSS file. This is the same array used in the ZOPEN call.
CPATH	Input	The pathname of the data to read. The pathname must meet the paired function data conventions.
NORD	Output	The number of ordinates (number of points per curve) read. Each curve within a single record will have the same number of ordinates.
NCURVE	Output	The number of curves retrieved in this record.
IHORIZ	Output	The variable number to appear on the horizontal axis for plotting (one for first variable, two for second).
C1UNIT	Output	The units of the first variable (e.g., 'FEET', 'PERCENT').
C1TYPE	Output	The type of data for the first variable. The following types are recognized by DSS utility programs:

UNT	Untransformed
LOG	Logarithmic - data expressed as logarithms.
PROB	Probability - data expressed in percent.

C2UNIT	Output	The units of the second variable.
C2TYPE	Output	The type of data for the second variable.
VALUES	Output	The data values retrieved. The first NORD elements in VALUES correspond to the first variable (i.e., the X axis values of the data points). The data for the second variable (the Y axis values) begins at element NORD+1. Y axis values for a second curve would begin at (NORD * 2) + 1.
KVALS	Input	The dimension of array VALUES. VALUES must be dimensioned to at least: KVALS = (NCURVE + 1) * NORD
NVALS	Output	The number of values retrieved.
CLABEL	Output	The labels for each curve. For example, if an ELEVATION-DAMAGE function is retrieved containing residential, agricultural and commercial damage, then CLABEL might be returned as: CLABEL(1) = 'RESIDENTIAL ' CLABEL(2) = 'AGRICULTURAL' CLABEL (3) = 'COMMERCIAL ' For this example, NCURVE would be returned with three, and CLABEL should be dimensioned to at least three.
KLABEL	Input	The dimension of CLABEL. No more than KLABEL labels will be placed into CLABEL. If you do not wish to retrieve any labels, set KLABEL to zero.
LABEL	Output	A logical variable indicating if labels were returned. LABEL will be set to .TRUE. if labels were retrieved (there will be NCURVE labels), otherwise it will be as .FALSE.
HEADU	Output	The optional user header array. This array usually may be decoded by subroutine ZUSTFH.
KHEADU	Input	The dimension of array HEADU. No more than KHEADU elements of the user header array will be retrieved. If you do not want to retrieve the user header, set this to zero.
NHEADU	Output	The number of elements in the user header actually retrieved. NHEADU will always be equal to or less than KHEADU.
ISTAT	Output	A status parameter indicating the success of the operation. If ISTAT is returned with zero, then the data was successfully read. The possible values are:

ISTAT	Description
0	All data retrieved.
-1	The record does not exist.
1	The dimension of VALUES (KVALS) was not large enough to retrieve all the data. Only KVALS values returned; the curves are incomplete.
20	The record is not paired data.

Remarks:

Up to fifty curves (with the same ordinates) can be stored in one record. The maximum number of labels is also fifty. Either all curves will have a label, or no curves will have labels. If the VALUES array is dimensioned smaller than the number of data values in the record, only the first KVALS values will be retrieved.

A debug trace will be printed when the message level (MLEVEL) is set to seven (or above) via subroutine ZSET.

Points can be located within a singly dimension array by the following example:

```
C       To print the data (as X, Y1, Y2, Y3, ...):
        DO 20 I=1,NORD
          WRITE (6,10) (VALUES(J),J=I,NVALS,NORD)
 10       FORMAT (' X:',F8.2,',  Y(s):',50(2X,F8.2))
 20     CONTINUE


C       To transform the data into a doubly dimensioned array:
        IPOS = 0
        DO 20 I=1,NCURVE+1
          DO 20 J=1,NORD
            IPOS = IPOS + 1
            CURVE(J,I) = VALUES(IPOS)
 20     CONTINUE
```

Example:

```
C       Retrieve Paired Data from a DSS file, then print it in the form:
C       1,  X, Y1, Y2, ...
C
        PARAMETER (KVALS=1000, KLABEL=50)
        INTEGER IFLTAB(600), NORD, NCURVE, IHORIZ, NVALS
        INTEGER NHEADU, ISTAT
        REAL VALUES(KVALS)
        CHARACTER CPATH*80, C1UNIT*8, C2UNIT*8, C1TYPE*8, C2TYPE*8
        CHARACTER CLABEL(KLABEL)*12, CNAME*64
        LOGICAL LABEL
```

```
C
C     Open the DSS file.
      CALL ZOPEN (IFLTAB, CNAME, ISTAT)
      IF (ISTAT.NE.0) GO TO 900
C
C     Get the pathname.
      CALL ZPATH (. . .
C
C     Retrieve the data.
      CALL ZRPD (IFLTAB, CPATH, NORD, NCURVE, IHORIZ,
     * C1UNIT, C1TYPE, C2UNIT, C2TYPE, VALUES, KVALS, NVALS,
     * CLABEL, KLABEL, LABEL, DUM, 0, NHEADU, ISTAT)
      IF (ISTAT.NE.0) GO TO 910
C
C     Write the record's pathname.
      CALL CHRLNB (CPATH, NPATH)
      WRITE (6,20) CPATH(1:NPATH)
 20   FORMAT (' Record Pathname: ',A)
C
C     Write the label information (if there are labels).
      IF ((LABEL).AND.(NCURVE.LE.KLABEL)) THEN
        WRITE (6,30)
 30     FORMAT (' Curve Labels:')
        DO 50 I=1,NCURVE
          WRITE (6,40) I, CLABEL(I)
 40       FORMAT (' Curve',I3,' Label: ',A)
 50     CONTINUE
      ENDIF
C
C     Write the data (as point, X, Y1, Y2, Y3, ...).
      DO 80 I=1,NORD
        WRITE (6,60) I, (VALUES(J),J=I,NVALS,NORD)
 60     FORMAT (' Point',I4,';  X:',F8.2,',  Y(s):',50(2X,F8.2))
 80   CONTINUE
C
```

Example results for an ELEVATION-DAMAGE function having two damage categories and eighteen ordinates:

Input:

```
CPATH = /JAMES RIVER/DR1/ELEVATION-DAMAGE//1980/PLAN B/
NPATH = 48
KLABEL = 10
KVALS = 1000
```

Output:

```
NORD = 18
NCURVE = 2
IHORIZ = 2
C1UNIT = 'FEET'
C1TYPE = 'UNT'
C2UNIT = '$1000'
C2TYPE = 'UNT'
NVALS = 54
LABEL = .TRUE.
CLABEL(1) = 'S.F. RES'
CLABEL(2) = 'COMMERCIAL'
ISTAT  = 0
```

The VALUES array contains all of the data:

VALUES(1) through VALUES(18) contain the ELEVATION data.
VALUES(19) through VALUES(36) contain DAMAGE data for "S.F. RES".
VALUES(37) through VALUES(54) contain DAMAGE data for "COMMERCIAL".

5.2 ZSPD - Store Paired Data

Purpose:

ZSPD stores paired (curve) data in a DSS file. Curve labels and the user header may be stored in addition to the data.

Calling Sequence:

```
     CALL ZSPD (IFLTAB, CPATH, NORD, NCURVE, IHORIZ,
   * C1UNIT, C1TYPE, C2UNIT, C2TYPE, VALUES,
   * CLABEL, LABEL, HEADU, NHEADU, IPLAN, ISTAT)
```

Declarations:

```
     INTEGER IFLTAB(600), NORD, NCURVE, IHORIZ
     INTEGER KVALS, KLABEL, NHEADU, IPLAN, ISTAT
     REAL VALUES(KVALS), HEADU(NHEADU)
     CHARACTER CPATH*80, CLABEL(KLABEL)*12
     CHARACTER C1UNIT*8, C1TYPE*8, C2UNIT*8, C2TYPE*8
     LOGICAL LABEL
```

Argument Description:

IFLTAB	Input/ Output	The DSS work space used to manage the DSS file. This is the same array used in the ZOPEN call.
CPATH	Input	The pathname of the data to store. The pathname must meet the paired function data conventions.
NORD	Input	The number of ordinates (number of points per curve). Each curve to be stored in a single record must have the same number of ordinates.
NCURVE	Input	The number of curves to store in this record.
IHORIZ	Input	The variable number to appear on the horizontal axis for plotting (one for first variable, two for second).
C1UNIT	Input	The units of the first variable (e.g., 'FEET', 'PERCENT').
C1TYPE	Input	The type of data for the first variable. The following types are recognized by DSS utility programs: UNT Untransformed LOG Logarithmic - data expressed as logarithms. PROB Probability - data expressed in percent.

C2UNIT	Input	The units of the second variable.
C2TYPE	Input	The type of data for the second variable.
VALUES	Input	The data values to store. The first NORD elements in VALUES correspond to the first variable (the X axis). The data for the second variable must begin at element NORD+1 (the Y axis). Y axis values for a second curve would begin at (NORD * 2) + 1.
CLABEL	Input	A optional character array with labels corresponding to each curve. For example, if an ELEVATION-DAMAGE function is to be stored containing residential, agricultural and commercial damage, then CLABEL might be as follows: CLABEL(1) = 'RESIDENTIAL ' CLABEL(2) = 'AGRICULTURAL' CLABEL(3) = 'COMMERCIAL ' For this example, NCURVE would be returned with three, and CLABEL should be dimensioned to at least three.
LABEL	Input	A logical variable indicating if labels are to be stored. LABEL must be set to .TRUE. to store labels, otherwise it should be set to .FALSE.
HEADU	Input	The optional user header array. Information should be placed in this array by subroutine ZSTFH. If no additional user information is to be stored, this may be a dummy argument and NHEADU should be set to zero.
NHEADU	Input	The number of elements in the user header array HEADU. If no header information is to be stored, set this to zero.
IPLAN	Input	A flag indicating whether to write over existing data or not:

IPLAN	**Description**
0	Always write the record to the file.
1	Only write the record if it is new (i.e., no record previously existed in that file under that pathname).
2	Only write the data if the record already existed in the file.

ISTAT	Output	A status parameter indicating the success of the operation. If ISTAT is returned with zero, then the data was successfully stored, otherwise an error occurred. The possible values are:

ISTAT	**Description**
0	The data was successfully stored.
-1	The IPLAN specified requested the record be written only if it was new, but the file already contained a record with the pathname supplied.

-2	The IPLAN specified requested the record be written only if it already existed, but the pathname supplied was not found.
-3	The pathname does not meet the paired data conventions.
-4	The number of ordinates is less than one.
-5	NCURVE is less than one or greater than fifty.

Remarks:

Up to fifty curves (with the same ordinates) can be stored in one record. The maximum number of labels is also fifty. Either all curves will have a label, or no curves will have labels.

A debug trace will be printed if the message level (MLEVEL) is set to 7 (or above) via subroutine ZSET.

Unless the number of data points for the curve(s) is known prior to obtaining them (for example, if you are reading them from an external file), the data usually must be read into a buffer, then reorganized into a singly dimensioned array before storing with ZSPD. Points can be converted from a doubly dimensioned array into a singly dimension array by the following example:

```
C
C     The data has been read into array CURVE as X, Y1, Y2, Y3, ...
      IPOS = 0
      DO 20 I=1,NCURVE+1
      DO 20 J=1,NORD
         IPOS = IPOS + 1
         VALUES(IPOS) = CURVE(J,I)
 20   CONTINUE
```

Example:

```
C     Read (a) Curve(s) from an external file, then store it in DSS.
C     Up to 10 curves (in one record) can be stored by this routine.
C     The external file contains data in the form:
C     X, Y1, Y2, . . .
C     X, Y1, Y2, . . .
C     END
C
      PARAMETER (KVALS=1000, KLABEL=10)
      INTEGER IFLTAB(600), NORD, NCURVE, IHORIZ
      INTEGER ISTAT, IBF(20), IEF(20), ILF(20)
      REAL VALUES(KVALS), CURVES(300,11)
      CHARACTER CPATH*80, C1UNIT*8, C2UNIT*8, C1TYPE*8, C2TYPE*8
      CHARACTER CLABEL(KLABEL)*12, CNAME*64, CLINE*80
```

```
C
C      Open the DSS file.
       CALL ZOPEN (IFLTAB, CNAME, ISTAT)
       IF (ISTAT.NE.0) GO TO 900
C
C      Get the pathname.
       CALL ZPATH (. . .
C
C      Get the number of Curves, IHORIZ.
       READ (5,*) NCURVE, IHORIZ
C
C      Get the data units and type.
       READ (5,20) C1UNIT, C1TYPE, C2UNIT, C2TYPE
 20    FORMAT ( ...
C
C      Read the label information.
       DO 40 I=1,NCURVE
         READ (5,30) CLABEL(I)
 30      FORMAT ( ...
 40    CONTINUE
C
C      Read the data (as X, Y1, Y2, Y3, ...).
       NORD = 0
 50    CONTINUE
       READ (5,60,END=200) CLINE
 60    FORMAT (A)
C
C      Did we reach the end of the data yet?
       IF (INDEX(CLINE,'END').GT.0) GO TO 100
C
C      Parse the line.
       CALL PARSLI (CLINE, 20, NFIELD, IBF, IEF, ILF)
       IF (NFIELD.NE.NCURVE+1) GO TO 900
C

C      Place the data in the curves array.
       NORD = NORD + 1
       DO 80 I=1,NFIELD
         CURVES(NORD,I) = XREAL (CLINE, IBF(I), ILF(I), IERR)
         IF (IERR.NE.0) GO TO 900
 80    CONTINUE
C
C      Go back and read the next value.
       GO TO 50
C
C
 100   CONTINUE
```

```
C       All the data has been read.  Transfer the data into
C       a singly dimensioned array.
        IPOS = 0
        DO 120 I=1,NCURVE+1
        DO 120 J=1,NORD
          IPOS = IPOS + 1
          VALUES(IPOS) = CURVES(J,I)
 120    CONTINUE
C
C       Store the data.
        CALL ZSPD (IFLTAB, CPATH, NORD, NCURVE, IHORIZ,
     *  C1UNIT, C1TYPE, C2UNIT, C2TYPE, VALUES, CLABEL, .TRUE.,
     *  HEADU, 0, 0, ISTAT)
        IF (ISTAT.NE.0) GO TO 910
C
C       . . .
```

Example results for storing an ELEVATION-DAMAGE function having two damage categories ("S.F. RES" and "COMMERCIAL") and five ordinates:

ELEVATION	**S.F. RES DAMAGE**	**COMMERCIAL DAMAGE**
500.0	0.0	0.0
502.0	25.8	0.0
504.0	51.2	323.4
506.0	93.8	655.7
508.0	137.9	809.1

Input:

```
CPATH = /JAMES RIVER/DR1/ELEVATION-DAMAGE//1980/PLAN B/
NORD = 5
NCURVE = 2
IHORIZ = 2
C1UNIT = 'FEET'
C1TYPE = 'UNT'
C2UNIT = '$1000'
C2TYPE = 'UNT'
LABEL = .TRUE.
CLABEL(1) = 'S.F. RES'
CLABEL(2) = 'COMMERCIAL'
NHEADU = 0
IPLAN  = 0
```

The VALUES array contains all of the data:

VALUES(1) through VALUES(18) contain the ELEVATION data.
VALUES(19) through VALUES(36) contain DAMAGE data for "S.F. RES".
VALUES(37) through VALUES(54) contain DAMAGE data for "COMMERCIAL".

For example:

```
  VALUES(1)  = 500.0
  VALUES(2)  = 502.0
  ...
  VALUES(5)  = 508.0
  VALUES(6)  =   0.0
  VALUES(7)  =  25.8
  ...
  VALUES(10) =  137.9
  VALUES(11) =    0.0
  VALUES(12) =    0.0
  ...
 VALUES(15) =  809.1
```

6 Text Subroutines

Text data is defined as generic alpha-numeric lines of text, where each line is preceded by a line feed character and ends with a carriage return character. It does not, at this time, accommodate other types of characters, such as those that would be used to create a graphical display. There are no definitive size limitations for a DSS text record, but it is recommended that a record contain no more than about 200 lines of text. There are no conventions set for the structure of a text record's pathname. However, it is recommended that the pathname parts be labeled in a descending order of importance, and that the pathname imply that the record contains text data and not one of the other types of data.

The maximum possible length of a line in a DSS text record is 160 characters. A reasonable maximum length is 132 characters. Text data is stored with trailing blanks (blanks to the right of the last character in a line) removed. A line feed character is stored at the beginning of each line, and a carriage return character is stored at the end. A blank line consists of a line feed, a single blank, and a carriage return. Generally, text is stored without carriage control (e.g., a blank in column 1), although this is up to the programmer, as DSS does not check for this.

The maximum number of bytes that can be stored in a text record is 9600 on DOS machines, and 16,000 on most other machines (including the line feed and carriage return). The number of lines that can be stored is dependent on the average length of the lines. If the average length is about forty characters, the maximum number of lines would be about 200 for DOS and about 350 for other computers. Generally, an appropriate number of lines to store in a single record is from two to 150 lines. If one desired to store a large amount of text, for example an entire book, the text could be divided into sections consisting of one to three pages. A sequence number (or page number) could be converted into character form (using subroutine INTGRC), and used as part of the pathname.

Subroutines ZRTEXT and ZRTXTA retrieve text records from a DSS file, and subroutines ZSTEXT and ZSTXTA store text data. ZRTEXT places the retrieved text in a file (or writes it on the screen), while ZRTXTA puts it into a character array. Conversely, ZSTEXT stores text read from an ASCII file, while ZSTXTA stores text from a character array.

6.1 ZRTEXT - Retrieve Text Data (Into a File)

Purpose:

Subroutine ZRTEXT retrieves text data from a DSS file, and places it into an ASCII file or writes it to the screen. The file or screen is identified by a unit number, which must have been opened by the calling program. An alternative subroutine, ZRTXTA, will retrieve text data and place the text into a character array.

Calling Sequence:

```
      CALL ZRTEXT (IFLTAB, CPATH, IUNIT, HEADU, KHEADU, NHEADU,
     * LCCNTL, NLINES, ISTAT)
```

Declarations:

```
      INTEGER IFLTAB(600), IUNIT, NLINES, ISTAT
      INTEGER KHEADU, NHEADU
      REAL HEADU(KHEADU)
      CHARACTER CPATH*80
      LOGICAL LCCNTL
```

Argument Description:

IFLTAB	Input/ Output	The DSS workspace used to manage the DSS file. This is the same array used in the ZOPEN call.
CPATH	Input	The pathname of the text record to retrieve.
IUNIT	Input	The unit number (connected to a file or the screen) of where to write the text data. The unit must be opened prior to calling ZRTEXT.
HEADU	Output	The optional user header array. This array usually may be decoded by subroutine ZUSTFH.
KHEADU	Input	The dimension of array HEADU. No more than KHEADU elements of the user header array will be retrieved. If you do not want to retrieve the user header, set this to zero.
NHEADU	Output	The number of elements in the user header actually retrieved. NHEADU will always be equal to or less than KHEADU.
LCCNTL	Input	A logical flag indicating whether a FORTRAN carriage control character (a blank space) should be placed at the beginning of each text line. If LCCNTL is set to .TRUE., a blank character will be inserted.

NLINES	Output	The number of lines in the text record.
ISTAT	Output	A status parameter indicating the success of the retrieval. If ISTAT is returned with zero, then the data was successfully retrieved. If ISTAT is any other value, no data was retrieved. The possible values are:

ISTAT	**Description**
0	The record was successfully retrieved.
-1	The record was not found.
-2	The record specified is not a text record.
-3	An internal buffer was not large enough to retrieve the record.
-4	An error occurred while writing the text data to IUNIT. The FORTRAN error code is printed in an error message.

Remarks:

ZRTEXT does not page the text (if the text is written to the screen, it will be displayed all at once). If an unknown amount of text is to be displayed on the screen, it is usually preferable that it first be written to a scratch file, then copied to the screen a page at a time.

Generally, text data is stored without carriage control characters (although DSS does not check for this). ZRTEXT uses FORTRAN to write each line of test to IUNIT. FORTRAN requires that the first character in a formatted write be used for carriage control. By setting LCCNTL to .TRUE. a blank space is inserted a the beginning of each line to account for this. Generally, when writing text data to the screen LCCNTL is set to .TRUE., and when writing to a file LCCNTL is set to .FALSE.

If an error occurs, an error message will be written to the output (MUNIT) if the message level (MLEVEL) is one or greater. The messages contain the pathname, and information about the error. A debug trace may be activated by setting the message level to seven (or above) with subroutine ZSET.

Example:

```
C     Read a text record from a DSS file.  If the output is to
C     the screen, display the text one page at a time.
C
      INTEGER IFLTAB(600)
      CHARACTER CPATH*80, CLINE*132, CNAME*64
      LOGICAL LSCREEN
C
      CALL ATTACH (5, 'INPUT', 'STDIN', 'S=O', CLINE, NSTAT)
      CALL ATTACH (6, 'OUTPUT', 'STDOUT', ' ', CLINE, ISTAT)
      CALL ATTACH (7, 'TEXT',   'STDOUT', ' ', CLINE, ISTAT)
```

```
       CALL ATTACH (8, 'SCRATCH', 'SCRATCH1', ' ', CLINE, ISTAT)
       CALL ATTEND
C
       CALL ZOPEN (IFLTAB, ...
       WRITE (6,*)'Enter Pathname'
       READ (5,20,END=800) CPATH
 20    FORMAT (A)
C
C      Is the text to be written to the screen or a file?
       INQUIRE (UNIT=7, NAME=CNAME)
       IF (CNAME(1:3).EQ.'CON') THEN
C        Text is to be written to the screen.  First write it to
C        a scratch file so that it can be paged to the screen.
         LSCREEN = .TRUE.
         IUNIT = 8
       ELSE
C        Text is to be written to a file.  No paging.
         LSCREEN = .FALSE.
         IUNIT = 7
       ENDIF
C
       CALL ZRTEXT (IFLTAB, CPATH, IUNIT, HEADU, 0, NHEADU,
    *  .FALSE., NLINES, ISTAT)
       IF (ISTAT.NE.0) GO TO 800
C
       IF (LSCREEN) THEN
C        Page text from the scratch file to the screen.
         REWIND (UNIT=IUNIT)
 40      CONTINUE
C
C        Write 24 lines of text to the screen.
         DO 80 I=1,24
            READ (IUNIT, 20, END=800) CLINE
            CALL CHRLNB (CLINE, N)
            IF (N.EQ.0) N = 1
            WRITE (7, 60) CLINE(1:N)
 60         FORMAT (1X,A)
 80      CONTINUE
C
C        Pause by waiting for a carriage return.
         READ (5,20) CLINE(1:1)
         GO TO 40
       ENDIF
C
 800   CONTINUE
       CLOSE (UNIT=8)
       CLOSE (UNIT=9)
```

6.2 ZRTXTA - Retrieve Text Data (Into an Array)

Purpose:

Subroutine ZRTXTA retrieves text data from a DSS file and places it into a character array. ZRTXTA is usually called when the text is to be processed by the calling program, or the amount of text retrieved is small enough that it conveniently fits into a character array. An alternative subroutine, ZRTEXT, retrieves text data and places it into a file.

Calling Sequence:

```
      CALL ZRTXTA (IFLTAB, CPATH, CARRAY, KLINES, NLINES,
     * HEADU, KHEADU, NHEADU, ISTAT)
```

Declarations:

```
      INTEGER IFLTAB(600), KLINES, NLINES, ISTAT
      INTEGER KHEADU, NHEADU
      REAL HEADU(KHEADU)
      CHARACTER CPATH*80, CARRAY(KLINES)*(*)
```

Argument Description:

IFLTAB	Input/ Output	The DSS workspace used to manage the DSS file. This is the same array used in the ZOPEN call.
CPATH	Input	The pathname of the text record to retrieve.
CARRAY	Input	A character array that will contain the text data. The first element of CARRAY (i.e., CARRAY(1)) will contain the first line of text.
KLINES	Input	The dimension (number of elements) of CARRAY. No more than KLINES lines of text will be placed into CARRAY. The length of CARRAY is implied by FORTRAN (e.g., CARRAY(KLINES)*80).
NLINES	Output	The number of lines retrieved and placed into CARRAY.
HEADU	Output	The optional user header array. This array usually may be decoded by subroutine ZUSTFH.
KHEADU	Input	The dimension of array HEADU. No more than KHEADU elements of the user header array will be retrieved. If you do not want to retrieve the user header, set this to zero.

NHEADU	Input	The number of elements in the user header actually retrieved. NHEADU will always be equal to or less than KHEADU.
ISTAT	Output	A status parameter indicating the success of the retrieval. If ISTAT is returned with zero, then the data was successfully retrieved. If ISTAT is negative, no data was retrieved. The possible values are:

ISTAT	**Description**
0	The record was successfully retrieved.
1	CARRAY was not large enough (the dimension KLINES) to hold all the text lines in the record. CARRAY will contain KLINES lines of text. No error message will be printed.
-1	The record was not found.
-2	The record specified is not a text record.
-3	An internal buffer was not large enough to retrieve the record.

Remarks:

Generally, ZRTXTA is called when the type of text to retrieve (its approximate length and number of lines) is known. Each element (line) in CARRAY is pre-blanked. A blank line in the text will show up as an element in CARRY containing all blanks. If a text line is longer than the length of CARRAY, it will be truncated to the length of CARRAY. (ISTAT will not reflect truncation; therefore CARRAY should be declared as long as necessary.) Elements in CARRAY greater than NLINES are undefined.

If an error occurs, an error message will be written to the output (MUNIT) if the message level (MLEVEL) is one or greater. The messages contain the pathname, and information about the error. No error message is printed if CARRAY is not large enough to hold all the lines of text in the record (ISTAT = 1). A debug trace may be activated by setting the message level to seven (or above) with subroutine ZSET.

Example:

```
C     Retrieve text data from a DSS file and write it to
C     a file connected to unit 9.  If the word "Location:"
C     is found in the text, pass the location name to the
C     subroutine PROCES.
C
      PARAMETER (KLINES=500)
      INTEGER IFLTAB(600)
      CHARACTER CPATH*80, CLOC*40, CARRAY(KLINES)*132
C
C
      CALL ZOPEN (IFLTAB, ...
C
      WRITE (6,*)'Enter Pathname'
      READ (5,20,END=800) CPATH
 20   FORMAT (A)
C
C
C     Retrieve the text data for this record.
      CALL ZRTXTA (IFLTAB, CPATH, CARRAY, KLINES, NLINES,
     * HEADU, 0, NHEADU, ISTAT)
      IF (ISTAT.LT.0) GO TO 800
C
C     Was the array large enough to hold the entire record?
      IF (ISTAT.EQ.1) THEN
        WRITE (6,*)'Caution:  Text record terminated at ',NLINES,' lines'
      ENDIF
C
C     Process the text, one line at a time.
      DO 60 I=1,NLINES
        CALL CHRLNB (CARRAY(I), NLEN)
C       Is this a blank line?  If so set its length to 1 for writing.
        IF (NLEN.EQ.0) NLEN = 1
C
C       Write the text line to unit 9.
        WRITE (9,40,ERR=900) CARRAY(I)(1:NLEN)
 40     FORMAT (A)
C
C       Scan for "Location:".  If found, process this location.
        J = INDEX (CARRAY(I)(1:NLEN), 'Location:')
        IF (J.GT.0) THEN
           CLOC = CARRAY(I)(J+9:)
           CALL PROCES (CLOC)
        ENDIF
C
 60   CONTINUE
C
```

6.3 ZSTEXT - Store Text Data (From a File)

Purpose:

ZSTEXT reads text from an ASCII file and stores it in a DSS text record. A unit number passed as an argument identifies the file. An alternative subroutine, ZSTXTA, will store text data from a character array.

Calling Sequence:

```
      CALL ZSTEXT (IFLTAB, CPATH, IUNIT, HEADU, NHEADU,
     *  NLINES, ISTAT)
```

Declarations:

```
      INTEGER IFLTAB(600), IUNIT, NLINES, ISTAT, NHEADU
      REAL HEADU(NHEADU)
      CHARACTER CPATH*80
```

Argument Description:

IFLTAB	Input/ Output	The DSS workspace used to manage the DSS file. This is the same array used in the ZOPEN call.
CPATH	Input	The pathname of the text record to store.
IUNIT	Input	The unit number of the file containing the text to store. The file must be opened (and rewound) prior to calling ZSTEXT.
HEADU	Input	The optional user header array. Information should be placed in this array by subroutine ZSTFH. If no user header is to be stored, this may be a dummy argument and NHEADU should be set to zero.
NHEADU	Input	The number of elements in the user header array HEADU. If no header information is to be stored, set this to zero.
NLINES	Output	The number of lines read from the file and stored in the text record.
ISTAT	Output	A status parameter indicating the success of the operation. If ISTAT is returned with zero, then the data was successfully stored. If ISTAT is any other value, no data was stored. The possible values are:

ISTAT	**Description**
0	The record was successfully stored.

ISTAT	**Description**
-3	An internal buffer was not large enough to store the record.
-4	An error occurred while reading the text data from the file. The FORTRAN error code will be printed.

Remarks:

ZSTEXT does not rewind the unit before reading. ZSTEXT can read from the keyboard (if IUNIT is connected to the keyboard).

ZSTEXT uses a FORTRAN read to read from IUNIT until the end-of-file is reached, or, alternatively, an end-of-file marker is found. The end-of-file condition (defined by an "END=" statement in a FORTRAN read) is typically used. The alternative end-of-file marker on Harris computers is a $EOF at the beginning of a new line, or a **CNTRL Z** at the beginning of a line for DOS and UNIX machines. Because these characters must be read by FORTRAN, a carriage return must follow. If IUNIT is connected to a keyboard, reading can only be terminated when an end-of-file marker is typed (the user must enter a **CNTRL Z** or $EOF, then a carriage return).

If an error occurs (ISTAT is not zero), an error message will be written to the output (MUNIT) if the message level (MLEVEL) is one or greater. The messages contain the pathname, and information about the error. A debug trace may be activated by setting the message level to seven (or above) with subroutine ZSET.

Example:

```
C       This program is initiated when a NWS weather summary is
C       received.  It calls a routine to retrieve the summary and
C       place it into a temporary scratch file.  ZSTEXT is then called
C       to store the summary in a DSS file.
C
        INTEGER IFLTAB(600)
        CHARACTER CTEMP*64, CNAME*64, CPATH*80
        CHARACTER CLOC*32, CDATE*8, CTIME*8
C
C
        CALL ATTACH ( 5, 'INPUT', 'STDIN', 'S=O', CTEMP, ISTAT)
        CALL ATTACH ( 6, 'OUTPUT', 'STDOUT', ' ', CTEMP, ISTAT)
        CALL ATTACH ( 0, 'DSSFILE', ' ', 'NOP', CNAME, ISTAT)
        CALL ATTACH ( 8, 'SCRATCH', 'SCRATCH2', ' ', CTEMP, ISTAT)
        CALL ATTEND
C
        CALL ZOPEN (IFLTAB, CNAME, ISTAT)
        IF (ISTAT.NE.0) STOP
C
C       LOADSM will retrieve the latest summary received from the NWS,
C       and place it in the file connected to unit 8.  The location,
C       date, and time, of the summary will also be returned.
        CALL LOADSM (8, CLOC, CDATE, CTIME, IERR)
        IF (IERR.NE.0) GO TO 900
C
        CALL ZPATH ('NWS', CLOC, 'SUMMARY', CDATE, CTIME, ' ', CPATH,
     *  NPATH)
C
        REWIND (UNIT=8)
        CALL ZSTEXT (IFLTAB, CPATH, 8, HEADU, 0, NLINES, ISTAT)
        IF (ISTAT.NE.0) GO TO 910
```

6.4 ZSTXTA - Store Text Data (From an Array)

Purpose:

Subroutine ZSTXTA stores ASCII text from a character array into a DSS text record. An alternative subroutine, ZSTEXT, stores text data from a file.

Calling Sequence:

```
     CALL ZSTXTA (IFLTAB, CPATH, CARRAY, NLINES, HEADU, NHEADU,
*    ISTAT)
```

Declarations:

```
     INTEGER IFLTAB(600), NLINES, ISTAT, NHEADU
     REAL HEADU(NHEADU)
     CHARACTER CPATH*80, CARRAY(NLINES)*(*)
```

Argument Description:

IFLTAB	Input/ Output	The DSS workspace used to manage the DSS file. This is the same array used in the ZOPEN call.
CPATH	Input	The pathname of the text record to store.
CARRAY	Input	A character array containing the text data to store. The array must be padded with blanks (i.e., characters to the right of the end of each line must be blanks). Trailing blanks are not stored. The length of CARRAY is implied by FORTRAN (e.g., CARRAY(NLINES)*132).
NLINES	Input	The number of lines in CARRAY to store.
HEADU	Input	The optional user header array. Information should be placed in this array by subroutine ZSTFH. If no user header is to be stored, this may be a dummy argument and NHEADU should be set to zero.
NHEADU	Input	The number of elements in the user header array HEADU. If no header information is to be stored, set this to zero.
ISTAT	Output	A status parameter indicating the success of the operation. If ISTAT is returned with zero, then the data was successfully stored. If ISTAT is any other value, no data was stored. The possible values are:

ISTAT	Description
0	The record was successfully stored.
-3	An internal buffer was not large enough to store the record.

Remarks:

If an error occurs (ISTAT is not zero), an error message will be written to the output (MUNIT) if the message level (MLEVEL) is one or greater. The messages contain the pathname, and information about the error. A debug trace may be activated by setting the message level to seven (or above) with subroutine ZSET.

Example:

```
C     A flood forecast has been made.  Read notes typed in by the
C     forecaster and store in a text record.
C
      INTEGER IFLTAB(600), IBF(2), IEF(2), ILF(2)
      CHARACTER CPATH*80, CBASIN*32, CDATE*9, CTIME*4, CLINE*80
      PARAMETER (KLINES=600)
      CHARACTER CARRAY(KLINES)*132
C
      CALL ZOPEN (IFLTAB, ...
C
      WRITE (6, *) 'Enter Forecast Basin Name'
      READ (5,20) CBASIN
 20   FORMAT (A)
C
      WRITE (6,*)'Enter Forecast Date and Time'
      READ (5,20) CLINE
      CALL PARSLI (CLINE, 2, NFIELD, IBF, IEF, ILF)
      IF (NFIELD.NE.2) GO TO 900
      CDATE = CLINE(IBF(1):IEF(1))
      CTIME = CLINE(IBF(2):IEF(2))
C
      WRITE (6,*)'Enter Forecast Notes.'
      WRITE (6,*)'Enter END at the beginning of the line when done.'
C
C     Read text from the forecaster.  Write data to DSS when "END" is
C     entered, or the number of lines reaches the dimension limit.
      NLINES = 0
 40   CONTINUE
        NLINES = NLINES + 1
        IF (NLINES.GT.KLINES) GO TO 100
        READ (5,20) CARRAY(NLINES)
        IF (CARRAY(NLINES)(1:3).EQ.'END') GO TO 100
      GO TO 40
```

```
C
 100  CONTINUE
      CALL ZPATH (CBASIN, 'FORECAST NOTES', CDATE, CTIME, ' ', ' ',
    * CPATH, NPATH)
C
C     Don't store the entry "END".
      NLINES = NLINES - 1
      CALL ZSTXTA (IFLTAB, CPATH, CARRAY, NLINES, HEADU, 0, ISTAT)
      IF (ISTAT.NE.0) GO TO 900
```

7 Catalog and Tag Subroutines

The following chapter describes the subroutines that are used to access or generate the catalog and condensed catalog, and the subroutines that utilize record tags.

The catalog file is a list of the record pathnames in a DSS file, along with their last written date and time and the name of the program that wrote that record. The catalog is usually sorted alphabetically by pathname parts. Each pathname has a record tag and a reference number, either of which may be used in place of the pathname in several of the utility programs. On most computers, the name given to the catalog file is the DSS file's name with a "C" appended to it (e.g., On UNIX "file.dssc"). On MS-DOS computers, the catalog file has an extension of ".DSC".

A catalog reference number is the sequential number of a pathname in the catalog file. These numbers are provided for quick interactive reference to a record from a utility program. When a number is given, the utility program sequentially searches the catalog file until it finds that number, and then returns the associated pathname. Reference numbers are temporary; they may change each time the catalog is updated.

A record tag is a one to eight character semi-permanent record identifier that is not necessarily unique. It must begin with a non-numeric valid tag character. Valid tag characters are the set of upper case characters, numbers, and the following characters: ! $ % & ( ) * + - . : ; < > ? [ ] { } \ | ~. The characters @ # can also be used, but are discouraged because they conflict with other uses. Invalid tag characters are the set of lower case characters, the space character, control characters (including the null character), the following characters: , ' " ` ^ / = and the "delete" character.

Tags can be set by the user, or can be set according to a scheme based on the parts of the pathname. For example, a scheme might cause a data record of observed flow with a B part of NATP to have a tag of NATP-OF. The default tag is the letter "T" followed by the "sequence number" of the record (the number of new records written to the file). Tag(s) may be used in place of pathnames in several DSS programs. If more than one pathname has the same tag, only the first one found will be used.

A special catalog, called a "condensed catalog", is useful primarily for time series data. In this type of catalog, pathname parts are listed in columns, and pathnames for time series data, which differ only by the date (D part), are referenced with just one line. Repeating parts are replaced by dashes for easier reading. On most computers, the name given to the condensed catalog file is the name of the DSS file with the letter "D" appended to it (e.g., On UNIX "file.dssd"). On MS-DOS computers, the condensed catalog file has an extension of ".DSD".

Subroutine ZOPNCA opens the catalog file, and determines if a valid catalog exists in that file. ZCAT generates a new catalog (and the optional condensed catalog), and can also obtain pathnames from a current catalog based on selective pathname parts (e.g., all pathnames with a "C" part of "FLOW"). A program may read pathnames from the catalog file with subroutine ZRDCAT, or read pathnames based on their reference number with ZRDPAT.

If a record's tag is known, ZTAGPA will retrieve its pathname using internal DSS tables, usually quicker than reading it from the catalog file. Subroutine ZSTAGS will set the tagging scheme to be used for new records stored in a DSS file. A single record tag may be changed by ZRETAG. All tags within a DSS file may be changed according the tagging scheme by subroutine ZRTALL.

Only utility programs usually use these subroutines. Two extensive examples of these subroutines are provided at the end of this section.

7.1 ZOPNCA - Open a Catalog File

Purpose:

ZOPNCA opens a DSS file's catalog file. If the catalog file does not exist, ZOPNCA can create it. If the file does exist, ZOPNCA returns the number of records in the catalog. ZOPNCA will also open the condensed catalog file, if desired.

Calling Sequence:

```
      CALL ZOPNCA (CDSSFI, ICUNIT, LGENCA, LOPNCA, LCATLG,
   *  ICDUNT, LGENCD, LOPNCD, LCATCD, NRECS)
```

Declarations:

```
      CHARACTER CDSSFI*64
      INTEGER ICUNIT, ICDUNT, NRECS
      LOGICAL LGENCA, LOPNCA, LCATLG, LGENCD, LOPNCD, LCATCD)
```

On MS-DOS microcomputers, NRECS must be INTEGER*4: INTEGER*4 NRECS

Argument Description:

CDSSFI	Input	The name of the DSS file whose catalog file is to be opened.
ICUNIT	Input	The unit number to open the catalog file with. (Most DSS utility programs use Unit 12 for the catalog file).
LGENCA	Input	A logical flag indicating whether the catalog file should be created if it does not exist. When set to .TRUE., the file will be created.
LOPNCA	Output	A logical variable indicating the status of the open. LOPNCA will be .TRUE. if the catalog file was successfully opened. If the file could not be opened, LOPNCA will be set to .FALSE.
LCATLG	Output	A logical variable returned as .TRUE. if the file contains a valid catalog. If LCATLG is .FALSE., ZCAT should be called to generate a new catalog of the DSS file.
ICDUNT	Input	The unit number to open the condensed catalog file with. (Most DSS utility programs use unit 13 for this file).
LGENCD	Input	A logical flag indicating whether the condensed catalog file should be created if it does not exist. When set to .TRUE., the file will be created.

LOPNCD	Output	A logical variable indicating the status of the condensed catalog open. LOPNCD will be .TRUE. if the condensed catalog file was successfully opened. If the file could not be opened, LOPNCD will be set to .FALSE.
LCATCD	Output	A logical variable returned as .TRUE. if the condensed catalog file contains a valid (condensed) catalog.
NRECS	Output	The number of records in the (regular) catalog file. This is the number shown in the catalog header.

Remarks:

Pathnames may be obtained from the catalog file with subroutine ZRDCAT, which is a general catalog reading routine, or by subroutine ZRDPAT, which obtains pathnames based on their reference number. If the tag of the pathname desired is known, subroutine ZTAGPA should be called to obtain the pathname.

The condensed catalog is designed primarily for DSS files with time series data (although it may be used with any type of record). It does not need to be accessed, but should be opened if it exists. If it does exist, it is typically updated whenever a new complete catalog is made (it does not require much additional computer time). The condensed catalog is for display purposes only; pathnames cannot be read from it.

A comprehensive example of ZOPNCA is provided at the end of this section.

Example:

```
      CHARACTER CDSSFI*64
      LOGICAL LOPNCA, LCATLG, LGENCD, LOPNCD, LCATCD
       INTEGER*4 NRECS
C
C     Open DSS file, etc.
C
C     Assume COPT contains command options.  IF COPT contains
C     a "C" (e.g., CA.C), get the condensed catalog.
      IF (INDEX(COPT,'C').GT.0) THEN
        LGENCD = .TRUE.
      ELSE
        LGENCD = .FALSE.
      ENDIF
C
C     Don't open the catalog if it is already opened (unless we
C     want a condensed catalog, and it is not yet opened).
```

```
      IF ((.NOT.LOPNCA).OR.(LGENCD.AND.(.NOT.LOPNCD))) THEN
        CALL ZOPNCA (CDSSFI, 12, .TRUE., LOPNCA, LCATLG, 13,
     *  LGENCD, LOPNCD, LCATCD, NRECS)
      ENDIF
C
      IF (.NOT.LOPNCA) GO TO 900
      IF (.NOT.LCATLG) CALL ZCAT (...
```

7.2 ZCAT - Catalog a DSS File

Purpose:

ZCAT generates a catalog (or listing) of the record pathnames in a DSS file. The catalog may be sorted by pathname parts. ZCAT can create a selective catalog by matching pathname parts. The selective catalog can be created from a current catalog (which is much more efficient than generating a new catalog), or directly from the DSS file. ZCAT can also produce an optional condensed catalog when generating a new catalog. The catalog files must be opened externally to ZCAT by subroutine ZOPNCA.

Calling Sequence:

```
      CALL ZCAT (IFLTAB, ICUNIT, ICDUNT, INUNIT, CINSTR,
     *  LABREV, LSORT, LCDCAT, NRECS)
```

Declarations:

```
      INTEGER IFLTAB(600), ICUNIT, ICDUNT, INUNIT, NRECS
      CHARACTER CINSTR*(*)
      LOGICAL LABREV, LSORT, LCDCAT
```

On MS-DOS microcomputers, NRECS must be INTEGER*4: INTEGER*4 NRECS

Argument Description:

IFLTAB	Input/ Output	The DSS workspace used to manage the DSS file. This is the same array used in the ZOPEN call.
ICUNIT	Input	The unit number of file where the catalog is to be written. If a new catalog is to be made, this should be the unit number of the catalog file. If a selective catalog is to be produced from an existing catalog, this unit should probably be attached to a scratch file.
ICDUNT	Input	The unit number of the condensed catalog file. If ICDUNT is set to zero, ZCAT will not generate a condensed catalog.
INUNIT	Input	The input catalog unit number. If a new catalog is to be made, this must be set to zero. If a selective catalog is to be produced from an existing catalog, this is the unit number of the existing catalog. If INUNIT is non-zero, the DSS file will not be cataloged.
CINSTR	Input	A character string containing any instructions for generating the catalog, such as the sort order or selective pathname parts.

		For example, if CINSTR is 'O=FB, C=FLOW', the catalog will be sorted in the pathname part order of FBACED, and only those pathnames with a C part of "FLOW" will be cataloged. CINSTR is usually a portion of the input line of the program. If no special instructions are given, set this to blank (' ').
LABREV	Input	A logical flag indicating whether an abbreviated catalog should be produced. If set to .TRUE., an abbreviated catalog will be generated, otherwise the standard catalog will be produced.
LSORT	Input	A logical flag indicating whether the pathnames should be sorted. When LSORT is set to .TRUE., the pathnames are sorted (this takes longer than an unsorted catalog).
LCDCAT	Output	A logical flag returned with .TRUE. if a condensed catalog was produced.
NRECS	Output	The number of records cataloged. This number will be the same as the reference number for the last pathname in the catalog file.

Remarks:

A description of the catalog and condensed catalog may be found in the "HECDSS User's Guide and Utility Program Manuals", Overview section. Additional information may also be found in the DSSUTL documentation, located in the same publication. Information about the selective catalog is located in Chapter 5 of the DSSUTL documentation.

The fastest catalog that can be generated is an unsorted abbreviated catalog. In this procedure, pathnames are just copied from the internal DSS address tables to the catalog. In a regular extended catalog, each record must be accessed to obtain the program name, date and time, etc.

After the catalog (or condensed catalog) has been generated, it may be displayed on the screen by reading directly from the catalog. ZCAT should not be used to display the catalog on the screen (do not set ICUNIT to standard output).

Pathnames may be read from the catalog file with subroutine ZRDCAT, which is a general catalog reading routine, or by subroutine ZRDPAT, which obtains pathnames based on their reference number. If you desire to display an abbreviated catalog and a regular catalog already exists, use subroutine ZRDPAT to read the pathnames from the catalog (see the example at the end of this section). The condensed catalog is for display purposes only; pathnames cannot be read from it.

DSS uses the computer's native sorting algorithm (not its own). On MS-DOS computers, the size of a file that can be sorted by the DOS sort function is 64,000 bytes. Thus, larger files will not be able to be sorted. (The typical limit for sorting is between 1,000 and 2,000 records.)

DOS also requires additional RAM for sorting. If this RAM is not available (a large program or other memory resident programs are present), the catalog will not be sorted.

Units 66, 67, 68, and 69 are used for sorting. On Harris computers, work files W2, W3, U1, and U2 are used for sorting (and their contents destroyed). On other computers the files dsssort.in, dsssort.out, and dsssort.tmp are used then deleted.

The condensed catalog is an optional catalog designed primarily for time series data (although it may be used with any record type). Generally, if the condensed catalog already exists, it should be updated when a complete sorted catalog is produced. A condensed catalog can only be made when the default sort order is used. (An abbreviated or complete catalog can be used, and selective pathname parts may be specified).

A status line with the percent complete can be displayed on the screen by setting 'CAST' to 'ON' with ZSET before calling ZCAT. The message unit (MUNIT) **must** be connected to the screen to display a status line.

A "catalog map" may be generated by ZCAT when creating a new catalog. A catalog map is a listing of the pathnames only (no title or reference numbers), which is useful for creating a input file of pathnames for some programs. This option is initiated by setting the map options in subroutine ZSET. The map file must be opened and its unit number passed to ZSET through the 'MAPUNT' parameter, then the ZSET 'MAP' parameter must be set to 'ON'. The catalog map is only created when a new catalog is generated. Be sure to call ZSET with 'MAP' set to 'OFF' after the map has been made.

A comprehensive example of ZCAT is provided at the end of this section.

Example 1:

```
      INTEGER IFLTAB(600)
      LOGICAL LCDCAT
      INTEGER*4 NRECS
C
C     Open the DSS file, etc.
C
C     Open the catalog file.
      CALL ZOPNCA (...
      IF (.NOT.LOPNCA) GO TO 900
C
C     If a condensed catalog may be made, set the unit number.
      IF (LOPNCD) THEN
        ICDUNT = 13
      ELSE
        ICDUNT = 0
      ENDIF
C
```

```
C       Generate a new catalog.
        CALL ZCAT (IFLTAB, 12, ICDUNT, 0, ' ', .FALSE., .TRUE.,
     *  LCDCAT, NRECS)
        IF (NRECS.EQ.0) GO TO 900
```

Example 2:

```
C       Search the DSS file for all pathnames
C       with a "B" part of "SOUTH BRIDGE".
        CHARACTER CPATH*80, CTAG*8
        INTEGER*4 NRECS
        LOGICAL LDUM
C
C       Open the DSS file.
        CALL ZOPEN ( ...
C
C       Use a scratch file for this catalog.
        OPEN (UNIT=12, STATUS='SCRATCH', ERR=900)
C
C       Call ZCAT to obtain a scratch catalog of pathnames
C        with a "B" part of "SOUTH BRIDGE".  Make the catalog
C       abbreviated and unsorted to get the pathnames quickly.
        CALL ZCAT (IFLTAB, 12, 0, 0, 'B=SOUTH BRIDGE', .TRUE.,
     *  .FALSE., LDUM, NRECS)
        IF (NRECS.EQ.0) GO TO 920
C
        Now process these pathnames.
        REWIND (UNIT=12)
 20     CONTINUE
C       Read a pathname.
        CALL ZRDCAT (12, .TRUE., 0, CTAG, IDUM, CPATH, NPATH, NFOUND)
C       Have we read all the pathnames?
        IF (NFOUND.EQ.0) GO TO 800
C
C       Process the pathname.
        CALL PROCES (CPATH)
C       Get the next one.
        GO TO 20
```

7.3 ZRDCAT - Read Pathnames from a Catalog File

Purpose:

ZRDCAT reads pathnames from a catalog file. All the pathnames in the catalog may be read, or pathnames may be obtained based on their record tags. ZRDCAT can return multiple pathnames in a single call, if desired.

Programs to obtain pathnames for computation or utility purposes frequently use this routine. If the pathnames are to be displayed on the screen, subroutine ZRDPAT is preferable. If the tag of a record is known, subroutine ZTAGPA will usually obtain its pathname faster than ZRDCAT can.

Calling Sequence:

```
      CALL ZRDCAT (ICUNIT, LALL, IOUNIT, CTAGS, NDIM,
     *   CPATHS, NPATHS, NFOUND)
```

Declarations:

```
      INTEGER ICUNIT, IOUNIT, NDIM, NPATHS(NDIM), NFOUND
      CHARACTER CTAGS(NDIM)*8, CPATHS(NDIM)*80
      LOGICAL LALL
```

Argument Description:

ICUNIT	Input	The unit number of the catalog file (from ZOPNCA).
LALL	Input	A logical variable that, when set to .TRUE., indicates all of the pathnames should be read from the catalog file. Under this condition, one pathname at a time is returned. NFOUND will be set to one until the end of the catalog is reached, at which point it will be set to zero. If LALL is .FALSE., then ZRDCAT will search for pathnames according to the tags given in CTAGS.
IOUNIT	Input	If desired, the pathnames can be written to a file instead of returned in the CPATHS array, when LALL is set to .FALSE.. IOUNIT is the unit number of this file (which must be opened prior to calling ZRDCAT). If the pathnames are to be returned in the variable CPATHS, set this to zero.
CTAGS	Input/ Output	If LALL is set to .TRUE., then CTAGS will be returned with the tag corresponding to the pathname read. If LALL is .FALSE., then CTAGS should be a character array (or a single character variable) containing the tags of the pathnames to read.

NDIM	Input	If LALL is .FALSE., this is the number of tags in CTAGS to search for, and must also be the dimension of CTAGS, CPATHS, and NPATHS. If LALL is set to .TRUE., this argument is ignored (only one pathname will be returned at a time).
CPATHS	Output	The pathname(s) retrieved. If LALL is .TRUE., this will contain the single pathname read from the catalog. If LALL is .FALSE., the first element of CPATHS will contain the pathname corresponding to the tag in the first element of CTAGS. If a pathname was not found for the corresponding tag, then that element of CPATHS will be blank filled. If IOUNIT is greater than zero, nothing will be returned in this argument.
NPATHS	Output	The length of the pathname(s) retrieved in CPATHS. If the pathname for the tag specified could not be found, then the corresponding element will be zero. NPATHS must be dimensioned to NDIM, regardless if IOUNIT is greater than zero or not, as it is used for internal bookkeeping. If LALL is .TRUE., NPATHS can be a single integer variable.
NFOUND	Output	The number of pathnames obtained. If LALL is .FALSE., then this is the number of pathnames returned in the array CPATHS. If LALL is .TRUE., then NFOUND will be set to one until the end of the catalog file is reached (and then it will be set to zero).

Remarks:

ZRDCAT is usually used when a program needs to read pathnames from the catalog file for processing. If pathnames are to be displayed on the screen, where reference numbers are used, subroutine ZRDPAT should be called instead. Although ZRDCAT will obtain pathnames based on their tags, subroutine ZTAGPA will obtain a pathname from a tag more efficiently than ZRDCAT can.

If several pathnames are to be searched for based on their tags, it is much more efficient to pass all the tags to ZRDCAT in the CTAGS array, than to call ZRDCAT once for each tag. If duplicate tags exist (for different pathnames), the pathname corresponding to the first matching tag is returned.

When IOUNIT is set greater than zero, ZRDCAT will write a sequence number (not the catalog reference number), the record tag, and the pathname. The format (I6,2X,A8,4X,A80) is used.

ZRDCAT rewinds the catalog before accessing it except for subsequent calls when LALL is .TRUE.. However, if LALL is .TRUE. and ZRDCAT does not reach the end of the catalog, the catalog file must be rewound before calling ZRDCAT again. (Call ZRDCAT until NFOUND returns 0, or rewind the catalog before using it the next time.)

Example 1:

```
C     Search the catalog file for pathnames
C      with a "B" part of "SOUTH BRIDGE".
      CHARACTER CPATH*80, CTAG*8
      INTEGER IBPART(6), IEPART(6), ILPART(6)
C
C     Open the DSS file, catalog file, etc.
C
 20   CONTINUE
C     Read a pathname.
      CALL ZRDCAT (12, .TRUE., 0, CTAG, IDUM, CPATH, NPATH, NFOUND)
C     Have we read all the pathnames from the catalog?
      IF (NFOUND.EQ.0) GO TO 800
C     Break it apart to get the B part.
      CALL ZUPATH (CPATH, IBPART, IEPART, ILPART, ISTAT)
      IF (ISTAT.NE.0) GO TO 900
C     Check the B part.
      IF (CPATH(IBPART(2):IEPART(2)).EQ.'SOUTH BRIDGE') THEN
        CALL PROCES (CPATH)
      ENDIF
      GO TO 20
```

Example 2:

```
C     We have 3 tags.  Obtain their pathnames from the catalog.
      CHARACTER CTAGS(3)*8, CPATHS(3)*80
      INTEGER NPATHS(3)
C
C     Open DSS file, catalog file, etc.
C
C     Set the CTAGS array.
      CTAGS(1) = 'T312'
      CTAGS(2) = 'SB-FLOW'
      CTAGS(3) = 'SB-PREC'
C
C     Read the pathnames.
      CALL ZRDCAT (12, .FALSE., 0, CTAGS, 3, CPATHS, NPATHS, NFOUND)
C
C     Print the pathnames.
      IF (NFOUND.EQ.0) THEN
        WRITE (6,*) 'No pathnames found matching the tags given.'
      ELSE
        DO 40 I=1,3
           IF (NPATHS(I).GT.0) WRITE (6,20)CTAGS(I),CPATHS(I)(1:NPATHS(I))
 20        FORMAT (' Tag: ',A,';  Pathname: ',A)
 40        CONTINUE
      ENDIF
```

Example 3:

```
C     Get tags from a command line, then obtain their pathnames from
C     the catalog and place them in a scratch file for later processing.
      CHARACTER CTAGS(20)*8, CDUM*1, CLINE*80
      INTEGER NPATHS(20), IBF(20), IEF(20), ILF(20)
C
C     Open DSS file, catalog file, etc.
C
C     Get the tags.
      WRITE (6,*)'Enter Record Tags'
      READ (5,20) CLINE
 20   FORMAT (A)
      CALL PARSLI (CLINE, 20, NFIELD, IBF, IEF, ILF)
      DO 40 I=1,NFIELD
        CTAGS(I) = CLINE(IBF(I):IEF(I))
 40   CONTINUE
C
C     Open a scratch file.
      OPEN (UNIT=15, STATUS='SCRATCH', ERR=900)
C
C     Read the pathnames.
      CALL ZRDCAT (12, .FALSE., 15, CTAGS, NFIELD, CDUM, NPATHS,
     * NFOUND)
C
      . . .


C     At some later point, get the pathnames and process.
      CHARACTER CTAG*8, CPATH*80
C
      REWIND (UNIT=15)
      DO 120 I=1,NFOUND
        READ (15, 110, END=800) J, CTAG, CPATH
 110    FORMAT (I6,2X,A8,4X,A)
        CALL PROCES (CTAG, CPATH)
 120  CONTINUE
```

7.4 ZRDPAT - Read a Pathname from a Catalog File by Reference Number

Purpose:

ZRDPAT searches for a pathname from a catalog file according to the pathname's reference number. ZRDPAT may be used in a loop to obtain a set of pathnames, or used to read a single pathname. If reference numbers are not used, subroutine ZRDCAT should be called instead. If the record tag is known, call subroutine ZTAGPA.

Calling Sequence:

```
CALL ZRDPAT (ICUNIT, IPOS, INUMB, CTAG, CPATH, NPATH, LEND)
```

Declarations:

```
INTEGER ICUNIT, IPOS, INUMB, NPATH
CHARACTER CPATH*80, CTAG*8
LOGICAL LEND
```

On MS-DOS microcomputers, the position variables must be INTEGER*4:

```
INTEGER*4 IPOS, INUMB
```

Argument Description:

ICUNIT	Input	The catalog unit number (from ZOPNCA).
IPOS	Input/ Output	A file position indicator used by ZRDPAT. When first reading from the catalog, ICUNIT should be rewound and IPOS set to zero. (The calling program must always set IPOS to zero when the catalog is rewound.)
INUMB	Input/ Output	The catalog reference number of the pathname to read. When the end of the catalog file is reached, INUMB will be returned with the reference number of the last pathname in the catalog. If INUMB is less than or equal to IPOS on input, ZRDPAT will return the next pathname and its reference number from the catalog.
CTAG	Output	The record tag of the pathname read.
CPATH	Output	The pathname corresponding to the reference number INUMB.
NPATH	Output	The number of characters in the pathname retrieved.

LEND	Output	A logical flag set to .TRUE. when the end of the catalog file has been reached. No pathname will be returned when LEND is .TRUE. (CPATH will be unchanged).

Remarks:

The catalog file must be rewound and IPOS set to zero before calling ZRDPAT to retrieve a (or set of) pathname(s). ZRDPAT only can search for pathnames in a forward direction; INUMB must always be greater than IPOS. Thus, if a sequence of reference numbers for pathnames to be retrieved is "12, 18, 9, 20", then the catalog has to be rewound and IPOS set to zero after reading pathname eighteen before pathname nine will be found. It is more efficient to sort the reference numbers in ascending order prior to calling ZRDPAT.

If INUMB is less than or equal to IPOS on input, ZRDPAT will read the next pathname in the catalog and return its reference number as INUMB. Thus, the entire catalog file can be read by rewinding the file, setting IPOS and INUMB both to zero, then calling ZRDPAT until LEND is .TRUE.. In this case INUMB does not need to be reset by the program each time ZRDPAT is called.

Example 1:

```
C     Obtain the pathname that has the reference number 24.
C
      CHARACTER CTAG*8, CPATH*80
      INTEGER*4 IPOS, INUMB
      LOGICAL LEND
C
      CALL ZOPNCA (...
C
      REWIND 12
      IPOS = 0
      INUMB = 24
      CALL ZRDPAT (12, IPOS, INUMB, CTAG, CPATH, NPATH, LEND)
C     If LEND is .TRUE., or INUMB is not 24, the pathname
C     was not found.
      IF ((LEND).OR.(INUMB.NE.24) GO TO 900
```

Example 2:

```
C     Read all the pathnames from the catalog.
C
      REWIND 12
      IPOS = 0
      INUMB = 0
10    CONTINUE
      CALL ZRDPAT (12, IPOS, INUMB, CTAG, CPATH, NPATH, LEND)
      IF (LEND) GO TO 900
```

```
         WRITE (6,20) INUMB, CTAG, CPATH(1:NPATH)
   20    FORMAT (1X,I6,2X,A,2X,A)
         GO TO 10
```

Example 3:

```
   C     Read the set of pathnames with the reference numbers of
   C     10 through 60.
   C
         REWIND 12
         IPOS = 0
         INUMB = 9
   10    CONTINUE
         INUMB = INUMB + 1
         CALL ZRDPAT (12, IPOS, INUMB, CTAG, CPATH, NPATH, LEND)
         IF (LEND) GO TO 900
         WRITE (6,20) INUMB, CTAG, CPATH(1:NPATH)
   20    FORMAT (1X,I6,2X,A,2X,A)
         IF (INUMB.LT.60) GO TO 10
```

Example 4:

```
   C     Read the set of pathnames whose reference numbers are
   C     contained in the array NUMBS (e.g., 8, 12, 15, 9, 20, 13).
   C
         REWIND 12
         IPOS = 0
         DO 20 I=1,JNUMBS
           INUMB = NUMBS(I)
   C       Are the numbers in ascending order?
   C       If not, rewind the catalog and reset IPOS.
           IF (INUMB.LE.IPOS) THEN
              REWIND 12
              IPOS = 0
           ENDIF
           CALL ZRDPAT (12, IPOS, INUMB, CTAG, CPATH, NPATH, LEND)
           IF (LEND) GO TO 900
           WRITE (6,10) INUMB, CTAG, CPATH(1:NPATH)
   10      FORMAT (1X,I6,2X,A,2X,A)
   20    CONTINUE
```

7.5 ZTAGPA - Get Pathnames from Tags

Purpose:

Given a record tag, or an array of tags, subroutine ZTAGPA obtains the corresponding pathname(s). ZTAGPA searches through tables in the DSS file, not the catalog file, to find the pathnames. ZTAGPA provides the fastest means of obtaining pathnames from tags.

Calling Sequence:

```
      CALL ZTAGPA (IFLTAB, IOUNIT, CTAGS, NDIM, CPATHS, NPATHS
   *  NFOUND)
```

Declarations:

```
      INTEGER IFLTAB(600), IOUNIT, NDIM, NPATHS(NDIM), NFOUND
      CHARACTER CTAGS(NDIM)*8, CPATHS(NDIM)*80
```

Argument Description:

IFLTAB	Input/ Output	The DSS workspace used to manage the DSS file. This is the same array used in the ZOPEN call.
IOUNIT	Input	If desired, the pathnames could be written to a file instead of returned in the CPATHS array. IOUNIT is the unit number of this file (which must be opened prior to calling ZTAGPA). If the pathname(s) are to be returned in variable CPATHS, set this to zero.
CTAGS	Input	A character variable or array containing the tag(s) of the pathnames to search for.
NDIM	Input	The number of tags in CTAGS to search for. CPATHS and NPATHS must have a minimum dimension of NDIM.
CPATHS	Output	The pathname(s) retrieved. If more than one tag is specified, then the first element of CPATHS will contain the pathname corresponding to the tag in the first element of CTAGS. If a pathname was not found for a tag, that element of CPATHS will be blank filled. If IOUNIT is greater than zero, this argument is ignored.
NPATHS	Output	The length of the pathname(s) retrieved in CPATHS. If the pathname for the tag specified could not be found, then this element will be zero. NPATHS must be dimensioned to

		NDIM, regardless if IOUNIT is greater than zero or not (it is used for internal bookkeeping).
NFOUND	Output	The number of pathnames retrieved. If a pathname is not found, this will be returned as zero.

Remarks:

If duplicate tags exist (the same tag is used for more than one pathname), the pathname for that the first matching tag found is returned. ZTAGPA has no capability of returning more than one pathname with an identical tag.

ZTAGPA first searches an internal tag table in the DSS file for matching tags. This table is automatically updated whenever the file is squeezed or cataloged. If the tag is not found in the internal table (the table has not been updated since the record was added), the main pathname-hash table will be searched. If the tag is in the tag table, the pathname will be retrieved very quickly. Otherwise the search will take somewhat longer. However, this is usually faster than searching the catalog file.

If several pathnames are to be retrieved based on their tags, it is more efficient to pass all the tags in the CTAGS array at one time than to call ZTAGPA for each tag.

When IOUNIT is greater than zero, ZTAGPA will write to that unit a sequence number (not the catalog reference number), the tag, and the pathname in the format (I6,2X,A8,4X,A80).

Example 1:

```
C     Get the pathname corresponding to the tag "LA-FLOW".
      CHARACTER CPATH*80, CTAG*8
C
C     Open the DSS file, etc.
C     CALL ZOPEN ( ...
C
      CTAG = 'LA-FLOW'
      CALL ZTAGPA (IFLTAB, 0, CTAG, 1, CPATH, NPATH, NFOUND)
C     Did we find it?
      IF (NFOUND.EQ.0) GO TO 800
C     Yes, process it.
      CALL PROCES (CPATH, NPATH)
```

Example 2:

```
C     We have 3 tags.  Print their pathnames.
      CHARACTER CTAGS(3)*8, CPATHS(3)*80
      INTEGER NPATHS(3)
C
C     Open the DSS file, etc.
```

```
C     CALL ZOPEN ( ...
C
C     Set the CTAGS array.
      CTAGS(1) = 'T312'
      CTAGS(2) = 'SB-FLOW'
      CTAGS(3) = 'SB-PREC'
C
C     Obtain the pathnames.
      CALL ZTAGPA (IFLTAB, 0, CTAGS, 3, CPATHS, NPATHS, NFOUND)
C     Print the pathnames.
      IF (NFOUND.EQ.0) THEN
        WRITE (6,*) 'No pathnames found matching the tags given.'
      ELSE
        DO 40 I=1,3
          IF (NPATHS(I).GT.0)WRITE (6,20)CTAGS(I),CPATHS(I)(1:NPATHS(I))
 20       FORMAT (' Tag: ',A,';  Pathname: ',A)
 40     CONTINUE
      ENDIF
```

Example 3:

```
C     Get tags from a command line.  Retrieve their pathnames
C     and place them in a scratch file for later processing.
      CHARACTER CTAGS(20)*8, CDUM*1, CLINE*80
      INTEGER NPATHS(20), IBF(20), IEF(20), ILF(20)
C
C     Open the DSS file, etc.
C     CALL ZOPEN ( ...
C
C     Get the tags.
      WRITE (6,*)'Enter Record Tags'
      READ (5,20) CLINE
 20   FORMAT (A)
      CALL PARSLI (CLINE, 20, NFIELD, IBF, IEF, ILF)
      DO 40 I=1,NFIELD
        CTAGS(I) = CLINE(IBF(I):IEF(I))
 40   CONTINUE
C
C     Open a scratch file.
      OPEN (UNIT=15, STATUS='SCRATCH', ERR=900)
C
C     Get the pathnames.
      CALL ZTAGPA (IFLTAB, 15, CTAGS, NFIELD, CDUM, NPATHS,
     * NFOUND)
C
      . . .
```

```
C       At some later point, get the pathnames and process.
C
        CHARACTER CTAG*8, CPATH*80
        REWIND (UNIT=15)
        DO 120 I=1,NFOUND
          READ (15, 110, END=800) J, CTAG, CPATH
 110      FORMAT (I6,2X,A8,4X,A)
          CALL PROCES (CTAG, CPATH)
 120    CONTINUE
```

7.6 ZRETAG - Change a Record Tag

Purpose:

Subroutine ZRETAG changes the tag of a single existing record. A new record's tag may be set by subroutine ZSET.

Calling Sequence:

```
CALL ZTAGPA (IFLTAB, CPATH, NPATH, CTAG, LFOUND)
```

Declarations:

```
INTEGER IFLTAB(600), NPATH
CHARACTER CPATH*80, CTAG*8
LOGICAL LFOUND
```

Argument Description:

IFLTAB	Input/ Output	The DSS workspace used to manage the DSS file. This is the same array used in the ZOPEN call.
CPATH	Input	The pathname of the record whose tag is to be changed.
NPATH	Input	The number of characters in CPATH.
CTAG	Input	The new tag for the record. The tag must be left justified and blank padded.
LFOUND	Output	A logical flag set to .TRUE. if the record exists and was retagged. If the record does not exist, LFOUND is set to .FALSE.

Remarks:

ZRETAG ignores any file tagging scheme, and sets the tag to that provided. The entire file may be re-tagged using the file's tagging scheme by calling subroutine ZRTALL.

Be sure that the new tag meets the tag requirements discussed at the beginning of this chapter.

7.7 ZSTAGS - Set the Tag Scheme for a DSS File

Purpose:

ZSTAGS sets or changes a DSS file's default tagging scheme. The scheme is set by a string that identifies the characters of pathname parts to make up the tags. A further description is given below in remarks.

Calling Sequence:

```
CALL ZSTAGS (IFLTAB, CSCHEM, ISTAT)
```

Declarations:

```
INTEGER IFLTAB(600), ISTAT
CHARACTER CSCHEM*(*)
```

Argument Description:

IFLTAB	Input/ Output	The DSS workspace used to manage the DSS file. This is the same array used in the ZOPEN call.
CSCHEM	Input	A character string containing the tagging scheme to be set (e.g., 'B1,B2,...'). To clear the file's tag scheme (so that sequence numbers will be used for tags), set CSCHEM to blank.
ISTAT	Output	A status parameter set to zero if no error occurred. If the tagging scheme was not recognized, this variable will be returned as -1.

Remarks:

A file tag scheme generates tags based on characters from pathnames. A typical tag might be the location name followed by part of the data parameter. For example, the observed flow at location NATP might have a tag of "NATP-FO"; the barometric pressure at FLD might be "FLD-BP". Each character in the tag scheme is set by specifying the pathname part letter (A, B, C, D, E, or F) followed by the character position number in that part. Tag character identifiers must be separated by commas. When a character or symbol without a character position is used, that character is inserted into the tag. For example, the tag "NATP-FO" may be generated by the following scheme:

B1,B2,B3,B4,-,C1,F1

This generates a tag using the first through fourth characters of the B part, a dash, the first character of the C part, then the first character of the F part. If no character corresponds to the position given, that character is ignored.

It is also possible to use characters from the second word of a part by preceding the part letter with an underscore "_". The tag "FLD-BP", from a pathname with a C part of "BAROMETRIC PRESSURE", was created using the following scheme:

B1,B2,B3,-,C1,_C1

The underscore causes the first character of the second word to be counted as position one. Words within a pathname are delimited by any of the following characters:

- @ _ + . ; : <blank>

If there is not a second word for a part, that tag character is ignored.

Only one tag scheme may be set for a file.

Example 1:

```
      INTEGER IFLTAB(600)
      CHARACTER CSCHEM*20
C
C     Open the DSS file, etc..
C
      CSCHEM = 'B1,B2,B3,B4,-,C1,_C1'
      CALL ZSTAGS (IFLTAB, CSCHEM, ISTAT)
C
      IF (ISTAT.NE.0) WRITE (6,20) CSCHEM
 20   FORMAT (' Invalid tagging scheme provided: ',A)
```

7.8 ZRTALL - Change All Record Tags in a DSS File

Purpose:

Subroutine ZRTALL changes all record tags in a DSS file according to the file's tag scheme (set by ZSTAGS). If no tag scheme is set, the records are re-tagged according to the current sequence number (e.g., T32).

Calling Sequence:

```
CALL ZRTALL (IFLTAB)
```

Declarations:

```
INTEGER IFLTAB(600)
```

Argument Description:

IFLTAB	Input/ Output	The DSS workspace used to manage the DSS file. This is the same array used in the ZOPEN call.

Remarks:

Subroutine ZSTAGS is usually called prior to ZRTALL.

7.9 Example of Obtaining Pathnames from References on a Command Line

The following comprehensive example takes a program's input line and obtains the pathnames identified by the user. The user may specify the pathname, a selective catalog reference (e.g., "C=FLOW"), catalog reference numbers, record tags, or "ALL". This example is a complete subroutine, which may be obtained on floppy diskette from the HEC.

```
      SUBROUTINE GETPAT (IFLTAB, CLINE)
C
C     Get pathnames for processing (call PROCES).
C
C     This subroutine takes a command line, figures
C     out what pathnames are referenced, then calls
C     subroutine PROCES with those paths (one at a time).
C     Example command lines are:
C     TA /OHIO/PITTSBURGH/FLOW/01JAN1950/1DAY/OBS/
C     TA 1, 29, 5-8, 3
C     TA C=FLOW F=OBS
C     TA T432, PITT-OF, T53
C     TA ALL
C
C     CLINE is the command line with the command removed.
C
C     Written by Bill Charley, HEC, 1990.
C
C
      PARAMETER (KTAGS=20)
      CHARACTER CLINE*(*), CPATH*80, CTAG*8, CDSSFI*64
      CHARACTER CPATHS(KTAGS)*80, CTAGS(KTAGS)*8
      INTEGER IFLTAB(*), NPATHS(KTAGS), IBF(20), IEF(20), ILF(20)
      INTEGER*4 IPOS, IBEG, IEND, ICOUNT, NRECS, IDUM
      LOGICAL LEND
C
      COMMON /LOGS/ LOPNCA, LCATLG, LOPNCD, LCATCD
      LOGICAL LOPNCA, LCATLG, LOPNCD, LCATCD
C
C
C     Get the length of the command line.
      CALL CHRLNB (CLINE, NLINE)
      IF (NLINE.EQ.0) GO TO 900
C
C
C     Was a regular pathname entered?
      IF (CLINE(1:1).EQ.'/') THEN
         CPATH = CLINE
         CALL PROCES (CPATH)
```

```
C
C          Was a reference to a catalog file made?
        ELSE IF ((INDEX(CLINE(1:5),'=').NE.0).OR.(CLINE(1:3).EQ.'ALL')
     *  .OR.(INDEX('123456789',CLINE(1:1)).GT.0)) THEN
C
C          Yes.  Be sure we have a catalog.
           IF (.NOT.LOPNCA) THEN
C             Get the name of the DSS file.
              CALL ZINQIR (IFLTAB, 'NAME', CDSSFI, IDUM)
              CALL ZOPNCA (CDSSFI, 12, .TRUE., LOPNCA, LCATLG, 13,
     *        .FALSE., LOPNCD, LCATCD, NRECS)
              IF (.NOT.LOPNCA) GO TO 910
C             If not cataloged, produce a new complete catalog.
              IF (.NOT.LCATLG) THEN
C                Is a condensed catalog associated with this DSS file?
                 IF (LOPNCD) THEN
                    ICDUNT = 13
                 ELSE
                    ICDUNT = 0
                 ENDIF
              CALL ZCAT (IFLTAB, 12, ICDUNT, 0, ' ', .FALSE.,
     *        .TRUE., LCATCD, NRECS)
              IF (NRECS.LE.0) GO TO 930
           ENDIF
        ENDIF
C
C       Check for a selective catalog reference.
        IF (INDEX(CLINE(1:5),'=').NE.0) THEN
C          Yes.  Open a scratch file to place pathnames in.
           OPEN (UNIT=14, STATUS='SCRATCH')
           REWIND 12
C          Have ZCAT make a selective catalog from the original catalog.
           CALL ZCAT (IFLTAB, 14, 0, 12, CLINE, .TRUE., .FALSE., LCATCD,
     *     NRECS)
           IF (NRECS.LE.0) GO TO 900
C          Read through the catalog.
 20        CONTINUE
           CALL ZRDCAT (14, .TRUE., 0, CTAG, 1, CPATH, NPATH, NFOUND)
C          All done?
           IF (NFOUND.EQ.0) THEN
              CLOSE (UNIT=14)
              GO TO 800
           ENDIF
           CALL PROCES (CPATH)
           GO TO 20
C
C          Were all pathnames specified?
```

```
          ELSE IF (CLINE(1:3).EQ.'ALL') THEN
C           Yes.  Read all pathnames from the catalog.
 40         CONTINUE
            CALL ZRDCAT (12, .TRUE., 0, CTAG, 1, CPATH, NPATH, NFOUND)
            IF (NFOUND.EQ.0) GO TO 800
            CALL PROCES (CPATH)
            GO TO 40
C           Was a catalog reference number given?
          ELSE IF (INDEX('123456789',CLINE(1:1)).GT.0) THEN
C           Yes.  Parse the command line.
            CALL PARSLI (CLINE, 20, NFIELD, IBF, IEF, ILF)
C
            REWIND 12
            IPOS = 0
C
C           Get the reference number(s).
            DO 80 I=1,NFIELD
C              Is there a dash between two numbers (e.g., "5-8").
               IDASH = INDEX (CLINE(IBF(I):IEF(I)),'-')
                  IF (IDASH.GT.0) THEN
C                    Yes.  Find the beginning and ending number
                     IBEG = INTGR (CLINE, IBF(I), IDASH-1, IERR)
                     IF (IERR.NE.0) GO TO 920
                     IEND = INTGR (CLINE, IBF(I)+IDASH, ILF(I)-IDASH, IERR)
                     IF (IERR.NE.0) GO TO 920
                     IF (IBEG.GT.IEND) GO TO 920
                  ELSE
C                    No a single number.  Make the end equal
C                    the beginning (e.g., 5-5).
                     IBEG = INTGR (CLINE, IBF(I), ILF(I), IERR)
                     IF (IERR.NE.0) GO TO 920
                     IEND = IBEG
                  ENDIF
C
C                 Now read the records.
                  IF (IBEG.LT.IPOS) THEN
                     REWIND 12
                     IPOS = 0
                  ENDIF
                  DO 60 ICOUNT=IBEG,IEND
                     CALL ZRDPAT (12, IPOS, ICOUNT, CTAG, CPATH, NPATH,
     .....*          LEND)
                     IF (LEND) THEN
                        WRITE (6,*)'The reference number given is greater than',
     *                  ' the number cataloged.'
                        GO TO 800
                     ENDIF
```

```
                CALL PROCES (CPATH)
 60           CONTINUE
 80         CONTINUE
C
          ENDIF
C
        ELSE
C
C         Must be Record Tags.
C         Parse the command line.
          CALL PARSLI (CLINE, 20, NFIELD, IBF, IEF, ILF)
C
          IF (NFIELD.GT.KTAGS) NFIELD = KTAGS
C         Copy the tags from the line into the CTAGS array.
          DO 100 I=1,NFIELD
            IF (ILF(I).GT.8) GO TO 950
            CTAGS(I) = CLINE(IBF(I):IEF(I))
 100      CONTINUE
C
C         Get the pathnames from the DSS file.
          CALL ZTAGPA (IFLTAB, 0, CTAGS, NFIELD, CPATHS, NPATHS,
     *    NFOUND)
C
C         Make sure that we found some pathnames.
          IF (NFOUND.EQ.0) GO TO 940
          IF (NFOUND.LT.NFIELD) THEN
            WRITE (6, 120)
 120        FORMAT (' ***  Unable to Find Pathnames for the',
     *      ' Following Tag(s):')
            DO 160 I=1,NFIELD
              IF (NPATHS(I).EQ.0) WRITE (6, 140) CTAGS(I)
 140          FORMAT (' Tag: ',A)
 160        CONTINUE
          ENDIF
C
          DO 180 I=1,NFIELD
            IF (NPATHS(I).GT.0) CALL PROCES (CPATHS(I))
 180      CONTINUE
C
        ENDIF
C
C
 800    CONTINUE
        RETURN
C
C
C       Error Messages.
```

```
 90   CONTINUE
      WRITE (6,*)'*** No Pathnames Match the Parameters Specified ***'
      GO TO 800
C
 910  CONTINUE
      WRITE (6,*)'*** Unable to access the catalog file ***'
      GO TO 800
C
 920  CONTINUE
      WRITE (6,*)'*** Unrecognizable Catalog Reference Number Given'
      GO TO 800
C
 930  CONTINUE
      WRITE (6,*)'**** No Records Cataloged - Possible Empty File ***'
      GO TO 800
C
 940  CONTINUE
      WRITE (6,*)' *** No Pathnames Match the Tag(s) Specified ***'
      GO TO 800
C
 950  CONTINUE
      WRITE (6,*)'*** Unrecognized Pathname Reference ***'
      GO TO 800
C
      END
```

7.10 Example of a Catalog Display Subroutine

The following is a comprehensive example of a subroutine that may be used by a program to process DSS catalog display requests. These requests include displaying the general catalog, the condensed catalog, and generating a new catalog with a variety of options. It may be used in conjunction with the previous example (subroutine GETPAT). This example is a complete subroutine, which may be obtained on floppy diskette from the HEC.

```
      SUBROUTINE CATALG (IFLTAB, CIN, COPT)
C
C     General DSS catalog functions.
C     Display catalog, unless options indicate otherwise.
C     On Input:
C     CIN - Input command line with command removed (e.g., "C=FLOW").
C     COPT - Command Options (parsed from command line):
C     A:  Abbreviated
C     C:  Condensed catalog
C     F:  Full (display all the catalog at once - don't pause)
C     M:  Catalog Map file
C     N:  Generate New catalog
C     P:  Send catalog to the printer
C     S:  Suppress catalog output
C     U:  Generate unsorted catalog (when new)
C
C     Written by Bill Charley, HEC, 1990
C
C
      CHARACTER CIN*(*), COPT*(*), CLINE*132, CPATH*80, CTAG*8
      CHARACTER CDSSFI*64
      INTEGER IFLTAB(*)
      LOGICAL LSORT, LABBR, LGENCD, LEND
      INTEGER*4 ICUR, IBEG, NREC, NORECS
C
      COMMON /LOGS/ LOPNCA, LCATLG, LOPNCD, LCATCD
      LOGICAL LOPNCA, LCATLG, LOPNCD, LCATCD
C
C
C
      ICUNIT = 12
      ICDUNT = 13
C
C     Check if the catalog file has been opened yet, or a
C     condensed catalog is requested and is unopened.
      IF ((.NOT.LOPNCA).OR.
     * ((.NOT.LOPNCD).AND.(INDEX(COPT,'C').GT.0))) THEN
C
```

```
C        Condensed catalog requested?
         IF (INDEX(COPT,'C').GT.0) THEN
            LGENCD = .TRUE.
         ELSE
            LGENCD = .FALSE.
         ENDIF
C        Get the name of the DSS file.
         CALL ZINQIR (IFLTAB, 'NAME', CDSSFI, IDUM)
C        Open the catalog file(s).
         CALL ZOPNCA (CDSSFI, ICUNIT, .TRUE., LOPNCA, LCATLG,
     *   ICDUNT, LGENCD, LOPNCD, LCATCD, NORECS)
C
         IF (.NOT.LOPNCA) THEN
            WRITE (6,*) ' *** Unable to Access the Catalog File ***'
            GO TO 820
         ENDIF
C
       ENDIF
C
C
C      Are we creating a new catalog?
       IF (INDEX(COPT,'N').GT.0) LCATLG = .FALSE.
C      Are we requesting a condensed catalog, and it does not exist?
       IF ((.NOT.LCATCD).AND.(INDEX(COPT,'C').GT.0).AND.
     * (LOPNCD)) LCATLG = .FALSE.
C
C      Should the Catalog file be updated?
       IF (LCATLG) THEN
         CALL ZINQIR (IFLTAB, 'NREC', CLINE, NREC)
         IF (NREC.NE.NORECS) THEN
            WRITE (6,20) NREC, NORECS
 20         FORMAT (/,' The Catalog File Needs to be Updated -',/,
     *      ' Current Number of Records in the DSS File:',I5,/,
     *      ' Number of Records in the Catalog File:    ',I5,/)
         ENDIF
       ENDIF
C
C      New option specified?
       IF (INDEX(COPT,'N').GT.0) LCATLG = .FALSE.
       IF ((.NOT.LCATCD).AND.(INDEX(COPT,'C').GT.0).AND.(LOPNCD))
     * LCATLG = .FALSE.
C
       REWIND ICUNIT
       IF (LOPNCD) REWIND ICDUNT
C
C      Should a new catalog be made?
```

```
          IF (.NOT.LCATLG) THEN
C
C            Check for 'Map' option.
             IF (INDEX(COPT,'M').GT.0) THEN
                OPEN (UNIT=MAPUNT, FILE='T3', IOSTAT=ISTAT)
                IF (ISTAT.EQ.0) THEN
                   WRITE (6,*)' Catalog Map file = T3'
                   CALL ZSET ('MAP', 'ON', I)
                   CALL ZSET ('MAPUNT', ' ', MAPUNT)
                ELSE
                   WRITE (6,*)' Unable to Access the Catalog Map File T3.'
                ENDIF
             ENDIF
C            Check for 'Unsorted' option.
             IF (INDEX(COPT,'U').GT.0) THEN
                LSORT = .FALSE.
                IF (INDEX(COPT,'C').GT.0) THEN
                   WRITE (6,*)' *** Must Generate a SORTED catalog to produce',
     *             ' the condensed version.'
                   LSORT = .TRUE.
                ENDIF
             ELSE
                LSORT = .TRUE.
             ENDIF
C
C            Check for 'Abbreviated' option.
             IF (INDEX(COPT,'A').GT.0) THEN
                LABBR = .TRUE.
             ELSE
                LABBR = .FALSE.
             ENDIF
C
C            Should we get a condensed catalog?
             IF (.NOT.LOPNCD) ICDUNT = 0
C
C            Now catalog the file.
             CALL ZCAT (IFLTAB, ICUNIT, ICDUNT, 0, CIN, LABBR, LSORT,
     *       LCATCD, NORECS)
C
             IF (INDEX(COPT,'M').GT.0) CLOSE (UNIT=MAPUNT)
C
C            Successful catalog ?
             IF (NORECS.LE.0) THEN
                IF (INDEX(CIN(1:5),'=').GT.0) THEN
                   WRITE (6,*)'*** No Pathnames Match Parts Specified ***'
                ELSE
                   WRITE (6,*)'*** No Records Cataloged: Empty File ***'
                ENDIF
          GO TO 820
          ENDIF
C
        LCATLG = .TRUE.
C
        ENDIF
C
        IF (INDEX(COPT,'S').GT.0) GO TO 800
C
C       Now display the catalog file.
        IUNIT = ICUNIT
C       Do we want to look at the condensed catalog?
        IF ((LCATCD).AND.(INDEX(COPT,'C').GT.0)) IUNIT = ICDUNT
        REWIND IUNIT
C       Set the number of lines to print on the screen at 22.
        JLINE = 22
C       Check for 'Full' option.
        IF (INDEX(COPT,'F').GT.0) JLINE = 30000

        ICUR = 0
        IBEG = 0
 100    CONTINUE
C
C       Check for abbreviated mode.
        IF ((INDEX(COPT,'A').GT.0).AND.(INDEX(COPT,'C').EQ.0)) THEN
C
          DO 140 I=1,JLINE
            CALL ZRDPAT (IUNIT, ICUR, IBEG, CTAG, CPATH, NPATH, LEND)
            IF (LEND) GO TO 800
            WRITE (6,120) ICUR, CTAG, CPATH(1:NPATH)
 120        FORMAT (1X,I6,2X,A,2X,A)
 140      CONTINUE
C
        ELSE
C         Long form, or condensed catalog.
          DO 180 I=1,JLINE
            READ (IUNIT, 150, END=800) CLINE
 150        FORMAT (A)
            CALL CHRLNB (CLINE, NLAST)
            IF (NLAST.EQ.0) NLAST = 1
            WRITE (6, 160) CLINE(1:NLAST)
 160        FORMAT (1X,A)
 180      CONTINUE
C
         ENDIF
C
C       Prompt user for next Screen of catalog or next command.
```

```
      IF (ICUR.LT.NORECS) THEN
        WRITE (6,*) 'Press Carriage Return To Continue,',
     *  ' or Enter New Command.'
        READ (5,200) CIN
 200    FORMAT (A)
        CALL CHRLNB (CIN, NIN)
        IF (NIN.EQ.0) GO TO 100
      ENDIF
C
 800  CONTINUE
C     Done displaying pathnames - print file if option given.
      IF (INDEX(COPT,'P').GT.0) CALL PRINTF (IUNIT, 'Catalog')
C
 820  CONTINUE
      RETURN
C
      END
```

8 General Read/Write Subroutines

The following chapter describes the DSS subroutines for reading or writing individual records. These subroutines should only be used for data that does not meet any of the standard DSS conventions (time series data, paired data, or text data). Data stored by one of these subroutines will not be recognized as a standard data type by the DSS programs. However, the data may be tabulated by DSSUTL.

A pathname does not have to follow the DSS conventions, although it cannot be more than eighty characters or less than four characters long. However, if it does not contain six parts separated by slashes, DSSUTL will not be able to access the record.

The basic routine for reading a record is ZREAD, and for writing a record is ZWRITE. These routines store or retrieve the user header array and the data array. The lengths of these arrays are treated as short integer words for several computers (e.g., HARRIS, MS-DOS). If floating point numbers are stored, then the array lengths need to be multiplied by the number of short integer words per real word. On UNIX workstations, all lengths are in long integer words (INTEGER*4).

ZREADX and ZWRITX are extended versions of ZREAD and ZWRITE. These subroutines will read or write an internal header array and the data compression array as well as the user header and data arrays. The arrays in ZREADX and ZWRITX are all in long integer words. There is more control with these subroutines than with ZREAD or ZWRITE.

ZRDBUF and ZWRBUF can perform "buffered" reading or writing. These subroutines can read or write a large amount of data in a single record with a relatively small data array. This is accomplished by calling the subroutine multiple times with the same pathname to store or retrieve the record. For example, 10,000 data values can be stored 500 at a time by calling ZWRBUF twenty times. ZWRBUF has the additional capability of being able to store data whose total number is unknown until the last call is made. For example, a program can read data from an external file into a relatively small array, then call ZWRBUF when that array becomes full, instead of having to read all the data into a large array and count the number of data values before storing it.

8.1 ZREAD - Read an Individual Record

Purpose:

ZREAD reads an individual record from a DSS file. It should be called only for data that does not follow the standard DSS conventions. ZREAD returns the user header and the data array as short integer words for Harris computers and MS-DOS computers. On other computers (e.g., UNIX) these arrays are in long integer words (INTEGER*4).

Calling Sequence:

```
      CALL ZREAD (IFLTAB, CPATH, NPATH, IHEADU, NHEADU, IDATA,
   *  NDATA, IPLAN, LFOUND)
```

Declarations:

```
      INTEGER NHEADU, NDATA, NPATH, IPLAN
      INTEGER IFLTAB(600), IHEADU(NHEADU), IDATA(NDATA)
      CHARACTER CPATH*80
      LOGICAL LFOUND
```

Argument Description:

IFLTAB	Input/ Output	The DSS workspace used to manage the DSS file. This is the same array used in the ZOPEN call.
CPATH	Input	The pathname of the data to retrieve. This pathname does not have to follow the DSS conventions. However, it cannot be greater than eighty characters or less than four characters long.
NPATH	Input	The number of characters in CPATH. Trailing blanks must be excluded.
IHEADU	Output	The user header array.
NHEADU	Input/ Output	The number of elements returned in the user header array. On input, this should be the dimension of IHEADU. If IPLAN is set to one, then NHEADU will be returned with the actual number of elements read. To have ZREAD not retrieve the user header, set this variable to zero. NHEADU must be a variable.
IDATA	Output	The data array.
NDATA	Input/ Output	The number of elements in the data array. As input, this should be the dimension of IDATA. If IPLAN is set to one, then this

		will be returned with the number of data elements read. NDATA must be a variable.
IPLAN	Input	This argument indicates whether the NHEADU and NDATA variables should be updated to the actual number of values read. If set to one, NHEADU and NDATA will be updated. If set to any other value, NHEADU and NDATA will not be changed.
LFOUND	Output	A logical status variable indicating if the record was found or not. If LFOUND is returned .TRUE., then the record was retrieved. If LFOUND is returned .FALSE., then the record does not exists not found, and no data was retrieved.

Remarks:

The IDATA array may either be real or integer. The number of data elements returned (NDATA) is given in integer words. On some computers (e.g., HARRIS, MS-DOS), two integer words are required for each real word and NDATA must be modified to reflect this (if real numbers are retrieved). The ninth word of the HECLIB common block "WORDS" contains the number of single integer words per real word. By either multiplying or dividing NDATA by this value, machine transportable code for retrieving real data can be produced. An example using this common block follows.

Example:

```
C       Retrieve the record named "/DATA SET 5/",
C       (which contains floating point numbers).
        INTEGER IFLTAB(600), IHEADU(100)
        REAL RDATA(1000)
        LOGICAL LFOUND
C
        COMMON /WORDS/ IWORDS(10)
C
        CALL ZOPEN (...
C
        NDATA = 1000
        NHEADU = 100
C
C       The record contains real values, so change NDATA to
C       reflect this.  Element 9 in common block "WORDS"
C       contains the number of short integer words in a
C       long (real) word.
        NDATA = NDATA * IWORD(9)
C
        CALL ZREAD (IFLTAB, '/DATA SET 5/', 12, IHEADU, NHEADU,
      * RDATA, NDATA, 1, LFOUND)
      IF (.NOT.LFOUND) GO TO 900
C
C     Change NDATA back to reflect real words.
      NDATA = NDATA / IWORD(9)
```

8.2 ZREADX - Read an Individual Record (Extended)

Purpose:

ZREADX reads an individual record from a DSS file. It not only returns the data and user header arrays, but the internal header and the compression header as well. ZREADX should only be called for data that does not follow the standard DSS conventions.

Calling Sequence:

```
      CALL ZREADX (IFLTAB, CPATH, HEADI, KHEADI, NHEADI,
   *  HEADC, KHEADC, NHEADC, HEADU, KHEADU, NHEADU, DATA,
   *  KDATA, NDATA, IPLAN, LFOUND)
```

Declarations:

```
      INTEGER IFLTAB(600), KHEADI, NHEADI, KHEADC, NHEADC
      INTEGER KHEADU, NHEADU, KDATA, NDATA, IPLAN
      REAL HEADI(KHEADI), HEADC(KHEADC), HEADU(KHEADU),
DATA(KDATA)
      CHARACTER CPATH*80
      LOGICAL LFOUND
```

Argument Description:

IFLTAB	Input/ Output	The DSS workspace used to manage the DSS file. This is the same array used in the ZOPEN call.
CPATH	Input	The pathname of the data to retrieve. This pathname does not have to follow the DSS conventions. However, it cannot be greater than eighty characters or less than four characters long.
HEADI	Output	The internal header array. This array contains items such as the units of the data.
KHEADI	Input	The dimension of HEADI. No more than KHEADI elements will be retrieved.
NHEADI	Output	The number of elements returned in HEADI.
HEADC	Output	The data compression header array. This array contains internal information on how the data is compressed.
KHEADC	Input	The dimension of HEADC. No more than KHEADC elements will be retrieved.
NHEADC	Output	The number of elements returned in HEADC.

HEADU	Output	The user header array. This array usually may be decoded by subroutine ZUSTFH.
KHEADU	Input	The dimension of HEADU. No more than KHEADU elements will be retrieved.
NHEADU	Output	The number of elements returned in HEADU.
DATA	Output	The data retrieved.
KDATA	Input	The dimension of the array DATA. No more than KDATA values will be returned.
NDATA	Output	The number of elements returned in the data array.
IPLAN	Input	An internal DSS flag. Set to zero.
LFOUND	Output	A logical status variable indicating if the record was found or not. If LFOUND is returned .TRUE., then the record was retrieved. If LFOUND is returned .FALSE., then the record does not exist and no data was retrieved.

Remarks:

The header and data arrays passed to ZREADX are treated as real arrays. They can be integer arrays as well, but the number variables (KHEADI, NHEADI, etc.) must be modified to reflect this on some computers. The ninth word of the HECLIB common block "WORDS" contains the number of single integer words per real word. By dividing the number variable by this value, machine transportable code for retrieving short integer data can be produced.

8.3 ZRDBUF - Read an Individual Record in a Buffered Mode

Purpose:

ZRDBUF reads an individual record from a DSS file. It can, if desired, buffer the data by reading a portion of the record at a time. This is useful when a large amount of data is to be retrieved. The data array can be relatively small, and multiple calls to ZRDBUF will retrieve all the data. ZRDBUF should only be called for data that does not follow the standard DSS conventions.

Calling Sequence:

```
          CALL ZRDBUF (IFLTAB, CPATH, HEADU, KHEADU, NHEADU, DATA,
KDATA,
     *    NDATA, LEND, IPLAN, LFOUND)
```

Declarations:

```
          INTEGER IFLTAB(600), KHEADU, NHEADU, KDATA, NDATA, IPLAN
          REAL HEADU(KHEADU), DATA(KDATA)
          CHARACTER CPATH*80
          LOGICAL LEND, LFOUND
```

Argument Description:

IFLTAB	Input/ Output	The DSS work space used to manage the DSS file. This is the same array used in the ZOPEN call.
CPATH	Input	The pathname of the data to retrieve. This pathname does not have to follow the DSS conventions. However, it cannot be greater than eighty characters or less than four characters long.
HEADU	Output	The user header array. If the number of elements stored is greater than KHEADU, subsequent calls to ZRDBUF will read the remainder of HEADU.
KHEADU	Input	The dimension of the array HEADU. No more than KHEADU values will be returned at a time.
NHEADU	Output	The number of elements returned in the user header array.
DATA	Output	The data retrieved. If the number of elements stored is greater than KDATA, then subsequent calls to ZRDBUF will return the remainder of the data.
KDATA	Input	The dimension of the array DATA. No more than KDATA values will be returned at a time.

NDATA	Output	The number of elements returned in the data array.
LEND	Output	A logical variable that is returned .TRUE. if all the data (including HEADU) has been read for this record. If LEND is .FALSE., then subsequent call to ZRDBUF (with the same pathname) will return more data.
IPLAN	Input	An internal DSS flag. Set to zero.
LFOUND	Output	A logical status variable indicating if the record was found or not. If LFOUND is returned .TRUE., then the record was retrieved. If LFOUND is returned .FALSE., then the record does not exist.

Remarks:

ZRDBUF may be called for a standard read, or a buffered read. For a buffered read, ZRDBUF is called multiple times with the same pathname, until LEND is returned as .TRUE.. An example of this follows.

The header and data arrays passed to ZRDBUF are treated as real arrays. They can be integer arrays as well, but the number variables (KHEADU, NHEADU, KDATA, and NDATA) must be modified to reflect this on some computers. The ninth word of the HECLIB common block "WORDS" contains the number of single integer words per real word. By dividing the number variable by this value, machine transportable code for retrieving short integer data can be produced.

Example:

```
C      Retrieve data and print it.
C      If more data exists than the size of the data array, call
C      ZRDBUF several times to read all of it.
C
       PARAMETER (KDATA=500, KHEADU=100)
       INTEGER IFLTAB(600)
       REAL DATA(KDATA), HEADU(KHEADU)
       LOGICAL LEND, LFOUND
       CHARACTER CPATH*80
C
C      Open the DSS file, etc.
C
       WRITE (6,*)'Enter Pathname'
       READ (5,20) CPATH
C
 20    CONTINUE
       CALL ZRDBUF (IFLTAB, CPATH, HEADU, KHEADU, NHEADU, DATA,
     * KDATA, NDATA, LEND, 0, LFOUND)
```

```
C
      IF (.NOT.LFOUND) THEN
        WRITE (6,40) CPATH
 40     FORMAT (' *** Record Not Found ***',/,' Pathname: ',A)
        GO TO 900
      ENDIF
C
      DO 60 I=1,NHEADU
        WRITE (6,*) HEADU(I)
 60   CONTINUE
C
      DO 80 I=1,NDATA
        WRITE (6,*) DATA(I)
 80   CONTINUE
C
      IF (.NOT.LEND) GO TO 20
```

8.4 ZWRITE - Write an Individual Record

Purpose:

ZWRITE writes an individual record to a DSS file. ZWRITE should only be called for data that does not follow the standard DSS conventions. ZWRITE stores the user header and the data array as short integer words for Harris computers and MS-DOS computers. On other computers (e.g., UNIX) these arrays are in long integer words (INTEGER*4).

Calling Sequence:

```
      CALL ZWRITE (IFLTAB, CPATH, NPATH, IHEADU, NHEADU, IDATA,
    * NDATA, IPLAN, LFOUND)
```

Declarations:

```
      INTEGER NHEADU, NDATA, NPATH, IPLAN
      INTEGER IFLTAB(600), IHEADU(NHEADU), IDATA(NDATA)
      CHARACTER CPATH*80
      LOGICAL LFOUND
```

Argument Description:

IFLTAB	Input/ Output	The DSS work space used to manage the DSS file. This is the same array used in the ZOPEN call.
CPATH	Input	The pathname of the data to store. This pathname does not have to follow the DSS conventions. However, if it does not contain six parts separated by slashes, utility programs (e.g., DSSUTL) will not be able to access the record. The pathname cannot be greater than eighty characters or less than four characters long.
NPATH	Input	The number of characters in CPATH. Trailing blanks must be excluded.
IHEADU	Input	An array that contains any additional user information to store. Generally, subroutine ZSTFH is called to prepare this array.
NHEADU	Input	The number of elements in IHEADU (in integer words).
IDATA	Input	The data to store. This may be either an integer array or a real array.
NDATA	Input	The number of elements in the data array to store (in integer words). If the array is declared as REAL, this number should

		be multiplied by the ninth element in the common block words, as described under remarks.
IPLAN	Input	An argument indicating whether to write over existing data or not:

IPLAN	Description
0	Always write the record to the file.
1	Only write the record if it new (i.e., it does not currently exist).
2	Only write the data if the record already exists.

If IPLAN is set to 1 or 2, and that condition is not met, then an error message will be written to the output (provided the message level is set to two or greater).

LFOUND	Output	A logical status variable indicating if the record already existed. If LFOUND is returned .TRUE., then the record existed, and was written over (unless IPLAN was set to one). If LFOUND is returned .FALSE., then the record did not previously exist.

Remarks:

The data may be real or integer. The number of elements to store is in integer words. On some computers (e.g., Harris, MS-DOS), two integer words are required for each real word, and NDATA must be modified to reflect this. The ninth word of the HECLIB common block "WORDS" contains the number of single integer words per real word. By either multiplying or dividing NDATA by this value, machine transportable code for storing real data can be produced. An example using this common block follows.

When a record is written, ZWRITE issues a message containing the pathname and version number of the record if the message level is 3 or greater (the default is 3). If an error occurs (such as IPLAN indicates only a new record may be written, and the record already exists), then an error message will be given for a message level of 2 or greater. If a fatal error occurs (e.g., there is no more disk space left), then an error message will be written, regardless of the message level.

Example:

```
C     Store a record named "/DATA SET 5/", (containing REAL data).
      INTEGER IFLTAB(600)
      REAL RDATA(1000)
      LOGICAL LFOUND
C
      COMMON /WORDS/ IWORDS(10)
C
      . . .
      CALL ZOPEN (...
      . . .
C
C     Compute the data values (and place in array RDATA).
C     . . .
C
C     The record contains real values, so change NDATA to
C     reflect short integer words.  Word 9 in common block
C     "WORDS" contains the number of short (integer) words
C     in a long (real) word.
C
      NDATA = NDATA * IWORD(9)
C
      CALL ZWRITE (IFLTAB, '/DATA SET 5/', 12, IHEADU, 0,
     * RDATA, NDATA, 0, LFOUND)
```

8.5 ZWRITX - Write an Individual Record (Extended)

Purpose:

ZWRITX writes an individual record to a DSS file. It will not only store the data and the user header array, but will also store an internal header array and a data compression header array. ZWRITX should only be called for data that does not follow the standard DSS conventions.

Calling Sequence:

```
     CALL ZWRITX (IFLTAB, CPATH, NPATH, HEADI, NHEADI,
*    HEADC, NHEADC, HEADU, NHEADU, DATA, NDATA, ITYPE,
*    IPLAN, ISTAT, LFOUND)
```

Declarations:

```
     INTEGER IFLTAB(600), NPATH, NHEADI, NHEADC
     INTEGER NHEADU, NDATA, ITYPE, IPLAN, ISTAT
     REAL HEADI(NHEADI), HEADC(NHEADC), HEADU(NHEADU),
DATA(NDATA)
     CHARACTER CPATH*80
     LOGICAL LFOUND
```

Argument Description:

IFLTAB	Input/ Output	The DSS workspace used to manage the DSS file. This is the same array used in the ZOPEN call.
IOUNIT	Input	If desired, the pathnames could be written to a file instead of returned in the CPATHS array. IOUNIT is the unit number of this file (which must be opened prior to calling ZTAGPA). If the pathname(s) are to be returned in variable CPATHS, set this to zero.
CPATH	Input	The pathname of the data to store. This pathname does not have to follow the DSS conventions. However, if it does not contain six parts separated by slashes, utility programs (e.g., DSSUTL) will not be able to access the record. The pathname cannot be greater than eighty characters or less than four characters long.
NPATH	Input	) The number of characters in CPATH. Trailing blanks must be excluded.
HEADI	Input	The internal header array. This array usually contains the data units and similar information.

NHEADI	Input	The number of elements in HEADI.
HEADC	Input	The data compression array.
NHEADC	Input	The number of elements in HEADC.
HEADU	Input	The user header array. Generally, subroutine ZSTFH is called to prepare this array.
NHEADU	Input	The number of elements in HEADU.
DATA	Input	The data array.
NDATA	Input	The number of elements in the data array to store.
IPLAN	Input	An argument indicating whether to write over existing data or not:

IPLAN	**Description**
0	Always write the record to the file.
1	Only write the record if it new (i.e., it does not currently exist).
2	Only write the data if the record already exists.

If IPLAN is set to 1 or 2, and that condition is not met, then an error message will be written to the output (provided the message level is set to two or greater).

ISTAT	Output	A status parameter indicating the success of the operation. If ISTAT is returned with zero, then the data was successfully stored. The possible values are:

ISTAT	**Description**
0	Data stored.
-1	The record does not exist, and IPLAN was set to two (write over existing records only).
-2	The record already exists, and IPLAN was set to one (do not write over existing records).
-10	An invalid pathname was given.
-11	An invalid number of data values (NDATA) were given.
-12	The DSS file has read access only.

LFOUND	Output	A logical status variable indicating if the record already existed. If LFOUND is returned .TRUE., then the record existed, and was written over (unless IPLAN was set to one). If LFOUND is returned .FALSE., then the record did not previously exist.

Remarks:

The header and data arrays passed to ZWRITX are treated as real arrays. They can be integer arrays as well, but the number variables (NHEADI, etc.) must be modified to reflect this for some computers. The ninth word of the HECLIB common block "WORDS" contains the number of single integer words per real word. By multiplying the number variable by this value, machine transportable code for storing short integer data can be produced.

When a record is written, ZWRITX issues a message containing the pathname and version number of the record if the message level is three or greater (the default is three). If an error occurs (such as IPLAN indicates only a new record may be written, and the record already exists), then an error message will be given for a message level of two or greater. If a fatal error occurs (e.g., there is no more disk space left), then an error message will be written, regardless of the message level.

8.6 ZWRBUF - Write an Individual Record in a Buffered Mode

Purpose:

ZWRBUF writes an individual record to a DSS file. It can, if desired, buffer the data by storing a portion of it at a time. This is useful when a large amount of the data is stored. The data array supplied to ZWRBUF can be relatively small, and multiple calls to the subroutine will store all the data. ZWRBUF should only be called for data that does not follow the standard DSS conventions.

Calling Sequence:

```
          CALL ZWRBUF (IFLTAB, CPATH, HEADU, NHEADU, NTOTH, DATA,
NDATA,
        *  NTOTD, LEND)
```

Declarations:

```
INTEGER IFLTAB(600), NHEADU, NTOTH, NDATA, NTOTD
REAL HEADU(NHEADU), DATA(NDATA)
CHARACTER CPATH*80
LOGICAL LEND
```

Argument Description:

IFLTAB	Input/ Output	The DSS workspace used to manage the DSS file. This is the same array used in the ZOPEN call.
CPATH	Input	The pathname of the data to store. This pathname does not have to follow the DSS conventions. However, if it does not contain six parts separated by slashes, utility programs (e.g., DSSUTL) will not be able to access the record. The pathname cannot be greater than eighty characters or less than four characters long.
HEADU	Input	The user header array. Generally, subroutine ZSTFH is called to prepare this array.
NHEADU	Input	The number of elements in HEADU for this call. If the header array needs to be buffered, then this value will be less than NTOTH, and subsequent calls will store the remainder of the array.
NTOH	Input	The total number of elements to store in the header area. This is usually equal to NHEADU, but will be greater if the header is to be buffered.

DATA	Input	The data array.
NDATA	Input	The number of elements in DATA for this call. If the data array needs to be buffered, then this value will be less than NTOTD, and subsequent calls will store the remainder of the array.
NTOTD	Input	The total number of elements to store in the data area. If this is not a buffered write, then NTOTD will equal NDATA. If the write is buffered, then this will be the total number of data values to be stored. If the amount of data to store is unknown, then set this variable to -1, and set LEND to indicate when all of the data has been stored. See remarks for a further explanation.
LEND	Input	A logical flag indicating the end of the data set. If the total number of data values to store is unknown when ZWRBUF is first called (NTOTD set to -1), then LEND should be set to .FALSE. until ZWRBUF is called with the last set of data. If NTOTD is not minus one, then this argument is ignored.

Remarks:

ZWRBUF may be called for a standard single write, or a buffered write. For a buffered write, ZWRBUF is called multiple times with the same pathname, until all the data is stored. If the write is to be un-buffered, set NTOTD equal to NDATA.

ZWRBUF can store data where the number of data values to be stored is unknown. To use the subroutine in this fashion, set NTOTD to minus one and LEND to .FALSE., then call ZWRBUF as many times as needed to store the data. On the last call for that data set, set LEND to .TRUE.. When used this way, the data is always stored at the end of the file, which will produce inactive space if the record already existed. NTOTH must always be specified.

The header and data arrays passed to ZWRBUF are treated as real arrays. They can be integer arrays as well, but the number variables (KHEADU, etc.) must be modified to reflect this for some computers. The ninth word of the HECLIB common block "WORDS" contains the number of single integer words per real word. By multiplying the number variable by this value, machine transportable code for retrieving short integer data can be produced.

Example 1:

```
PARAMETER (KHEADU=100, KDATA=500)
INTEGER IFLTAB(600)
REAL HEADU(KHEADU), DATA(KDATA)
CHARACTER CPATH*80
LOGICAL LEND, LDUM
```

```
C
C     Open the DSS file, etc..
C
C
C     Store 400 values with ZWRBUF in a non-buffered write.
      READ (5,20) CPATH
 20   FORMAT (A)
C
C     Get the data values.
      CALL DATVAL ( DATA, HEADU)
      NHEADU = 50
      NDATA = 400
      CALL ZWRBUF (IFLTAB, CPATH, HEADU, NHEADU, NHEADU,
     * DATA, NDATA, NDATA, LDUM)
C
```

Example 2:

```
C     Store 10,000 values a buffered mode, where the total
C     number of data values is known.
      READ (5,20) CPATH
C
      NTOTD = 10000
      ICOUNT = 0
C
      DO 100 I=1,10000
        ICOUNT = ICOUNT + 1
C       Get a data value.
        CALL GETVAL (DATA(ICOUNT))
C       If we have reached the array dimension limit, store the data.
        IF (ICOUNT.EQ.KDATA) THEN
          CALL ZWRBUF (IFLTAB, CPATH, HEADU, 0, 0,
     *    DATA, ICOUNT, NTOTD, LDUM)
C         Reset the counter.
          ICOUNT = 0
        ENDIF
 100  CONTINUE
C
```

Example 3:

```
C     Store data in a buffered mode, where the total
C     number of data values to store is unknown.
      READ (5,20) CPATH
C
```

```
C     We don't know how much to store (but LEND will be set
C     to .TRUE. when we have all the data).
      NTOTD = -1
      ICOUNT = 0
C
 200  CONTINUE
      ICOUNT = ICOUNT + 1
C     Get a data value.
      CALL GETNUM (DATA(ICOUNT), LEND)
C     If we have reached the array dimension limit, or the last
C     data value has been calculated, store the data array.
      IF ((ICOUNT.EQ.KDATA).OR.(LEND)) THEN
        CALL ZWRBUF (IFLTAB, CPATH, HEADU, 0, 0,
     *  DATA, ICOUNT, NTOTD, LEND)
C       Do we need to compute more data?
        IF (LEND) GO TO 300
C       Reset the counter.
        ICOUNT = 0
      ENDIF
C
      GO TO 200
C
C
 300  CONTINUE
```

9 Utility Subroutines

The following chapter describes several DSS utility subroutines. Some of these routines (e.g., ZSTFH) are for use by general application programs, while others (e.g., ZRENAM) are usually used only by utility programs.

Subroutine ZSTFH places additional record information in the user header array. Such information may include a project's latitude and longitude, or the operator's name. Rating tables often include a datum, shift, offset and transform in the user header. A program can decode information from the user header with ZUSFTH.

Subroutine ZCHECK determines if a record exists (similar to ZDTYPE), and returns the number of data values and user header elements stored. Subroutine ZRECIN will display information about a record, while ZFILST will display information about a file. These two subroutines are used for display purposes only; subroutine ZINQIR returns information about a record or file for a program's internal use.

Subroutine ZCOREC copies a record from one DSS file to another, or duplicates a record within the same file. ZCOFIL copies a DSS file to a new DSS file, or appends the file to an existing DSS file.

Subroutine ZDELET deletes a record from a DSS file by flagging a record status cell. The data is not physically removed until the file is squeezed by DSSUTL. The record may be recovered by calling subroutine ZUNDEL (until file is squeezed). All records in a DSS file that were deleted by ZDELET may be recovered by ZUDALL. ZUDALL can also display a list of the deleted records within the file that are recoverable.

A record can be renamed (its pathname changed) with subroutine ZRENAM. Information in the DSS file or in the IFLTAB array can be decoded with ZDEBUG. ZDEBUG is used only for "low level" de-bugging. Subroutine trace statements activated by setting the message level MLEVEL with ZSET are intended to be used for de-bugging a program interface with DSS.

9.1 ZSTFH - Stuff the User Header Array

Purpose:

ZSTFH places user information in the user header array in preparation for writing to a DSS record. The information is stored in Hollerith (alpha-numeric) format. Each header item is identified by a label indicating what the data is. ZSTFH places a colon (:) between each label and item, and a semi-colon (;) after each item. For example, if the item 1234.0 has the label "DATUM", and the item "LOGLOG" has the label "TRANSFORM", the following header array would be produced:

0032DATUM:1234.0; TRANSFORM:LOGLOG;

The first element in the header will contain the total number of bytes in the array (0032 in this example).

ZSTFH can stuff several items in the header at one time, or append information to the header (allowing multiple calls to ZSTFH to stuff multiple items).

A user header assembled by ZSTFH can be disassembled by subroutine ZUSTFH.

Calling Sequence:

```
        CALL ZSTFH (CLABEL, CITEM, NITEM, HEADU, KHEADU, NHEADU,
ISTAT)
```

Declarations:

```
        INTEGER NITEM, KHEADU, NHEADU, ISTAT
        CHARACTER CLABEL(NITEM)*(*), CITEM(NITEM)*(*)
        REAL HEADU(KHEADU)
```

Argument Description:

CLABEL	Input	A character string identifying the item to stuff. Usually this is a single word (e.g., "DATUM"), but it can be more than one (with embedded blanks). If more than one item is to be stored, then this must be a character array, and the first element of CLABEL corresponds to the first element of CITEM. A label may not contain a colon or semi-colon.
CITEM	Input	A character string containing the item to stuff. If the item is a number, it must be converted to character by an internal write or with subroutine INTGRC or XREALC. The item may contain blanks but not a colon or semi-colon. If more than one

		item is to be stored, then this must be a character array, and the first element of CITEM corresponds to the first element of CLABEL.
NITEM	Input	The number of items to store in this call, which is also the dimension of arrays CLABEL and CITEM. This is often set to one.
HEADU	Input/ Output	The header array to stuff.
KHEADU	Input	The dimension of HEADU. No more than KHEADU elements will be stored in HEADU.
NHEADU	Input/ Output	The number of elements in HEADU. On the first call set this to zero to initialize HEADU.
ISTAT	Output	A status parameter indicating the success of the operation. If ISTAT is returned with zero, then information was successfully added to the user header array. If ISTAT is non-zero, an error occurred. The possible values are:

ISTAT	**Description**
0	Successful operation.
1	A label contains all blanks.
2	An item contains all blanks.
3	The number of elements required for this information is greater than KHEADU. Increase the dimension of HEADU.
4	Array HEADU is invalid. This is most probably caused by not initializing HEADU for this data set (set NHEADU to zero on the first call).
5	NITEM is less than or equal to zero.

Remarks:

On the first call to ZSTFH set NHEADU to zero. Information in the header cannot be edited. The array would need to be completely un-stuffed, then re-stuffed.

The label and item variables can each be up to 60 characters in length. However, it is recommended to keep the combination of both under 70 characters for manageability.

Information stored in the user header does not have to be generated by ZSTFH, although if it is not, it cannot be displayed by DSS utility programs.

The minimum dimension of the header array can be computed from the number of bytes to be stuffed (the length of the labels and items) and the number of bytes per real word. The minimum dimension is:

[(number of bytes + (3 * number of items)) / bytes per word] + 2

Usually, a dimension size of fifty words is sufficient for most purposes

A debug trace is available by setting the message level to nine with ZSET.

Example 1:

```
C    Store information about a gage in the user header,
C    by appending information to the header.
     PARAMETER (KHEADU=100)
     REAL HEADU(KHEADU)
C
     NHEADU = 0
     CALL ZSTFH ('USGS GAGE ID', '012345678', 1, HEADU, KHEADU,
   * NHEADU, ISTAT)
     IF (ISTAT.NE.0) GO TO 900
     CALL ZSTFH ('LATITUDE', '412345', 1, HEADU, KHEADU,
   * NHEADU, ISTAT)
     IF (ISTAT.NE.0) GO TO 900
     CALL ZSTFH ('LONGITUDE', '1212345', 1, HEADU, KHEADU,
   * NHEADU, ISTAT)
     IF (ISTAT.NE.0) GO TO 900
     CALL ZSTFH ('DATUM', '1234.56', 1, HEADU, KHEADU,
   * NHEADU, ISTAT)
     IF (ISTAT.NE.0) GO TO 900
     CALL ZSTFH ('OPERATOR', 'JOHN DOE', 1, HEADU, KHEADU,
   * NHEADU, ISTAT)
     IF (ISTAT.NE.0) GO TO 900
```

Example 2:

```
C    Store information about a gage in the user header,
C    with one call to ZSTFH.
     PARAMETER (KHEADU=100)
     REAL HEADU(KHEADU)
     CHARACTER CLABEL(5)*20, CITEM(5)*20
C
     DATA CLABEL(1) /'USGS GAGE ID'/
     DATA CLABEL(2) /'LATITUDE'/
     DATA CLABEL(3) /'LONGITUDE'/
```

```
      DATA CLABEL(4) /'DATUM'/
      DATA CLABEL(5) /'OPERATOR'/
C
      CITEM(1) = '012345678'
      WRITE (CITEM(2), '(I7)') LAT
      WRITE (CITEM(3), '(I7)') LONG
      WRITE (CITEM(4), '(F10.2)') DATUM
      CITEM(5) = 'JOHN DOE'
      NHEADU = 0
      CALL ZSTFH (CLABEL, CITEM, 5, HEADU, KHEADU, NHEADU,
ISTAT)
      IF (ISTAT.NE.0) GO TO 900
```

Examples 1 and 2 would generate the following header array:

```
    0094USGS GAGE ID:012345678; LATITUDE:412345; LONGITUDE:1212345;
    DATUM:1234.56; OPERATOR:JOHN DOE;
```

9.2 ZUSTFH - Disassemble the User Header Array

Purpose:

ZUSTFH disassembles a user header array created by ZSTFH. ZUSTFH can either return all items in the array (returning one at a time), or search for specific items. For a description of this array, see the ZSTFH documentation.

Calling Sequence:

CALL ZUSTFH (CLABEL, CITEM, NITEM, IPOS, HEADU, NHEADU, ISTAT)

Declarations:

INTEGER NITEM, IPOS, NHEADU, ISTAT
CHARACTER CLABEL(NITEM)*(*), CITEM(NITEM)*(*)
REAL HEADU(NHEADU)

Argument Description:

CLABEL	Input/ Output	If specific items are to be searched for in the header, this is the labels of those items to search for. Labels must be left-justified and blank filled. If ZUSTFH is to return all items in the header (NITEM set to zero), one at a time, then this will be returned with the label corresponding to CITEM.
CITEM	Output	The items corresponding to CLABEL. If you are searching for specific items, and the label given was not found, then CITEM will be returned blank filled.
NITEM	Input	If ZUSTFH is to search for specific items in the array, then NITEM is the number of items to search for and CLABEL and CITEM must be dimensioned to at least NITEM. If all items in the header array are to be returned (one at a time), then this should be set to zero on the first call, and ZUSTFH should be called multiple times, returning one item at a time, until IPOS is returned with -1.
IPOS	Input/ Output	An internal position variable. If all items are to be returned from the header (NITEM set to 0), then this variable must be set to zero on the first call. When all items have been returned, then IPOS will be returned as -1. If ZUSTFH is to search for specific items, then this argument is ignored. IPOS must be a variable.
HEADU	Input	The user header array.

NHEADU	Input	The number of elements in HEADU.
ISTAT	Output	A status parameter indicating the success of the operation. If ISTAT is zero, then information was successfully returned from the user header array. If ISTAT is less than zero, an error occurred and no items were returned. If ISTAT is greater than zero, an error occurred and portions of the items are returned. The possible values are:

ISTAT	**Description**
0	Successful operation.
-1	The header array is invalid. (The first element of HEADU does not contain a valid header count.)
-2	The internal header count indicates that the header array is greater than the size passed (NHEADU).
2	The length of the item or label is greater than the character variable CITEM or CLABEL. The item or label is truncated to the variable length.
4	The length of the item or label is greater than sixty, an internal dimension limit. The item or label is truncated to sixty characters.

Remarks:

The items and labels are returned left-justified, blank filled, with colons and semi-colons removed. There is no debug trace for ZUSTFH.

Examples:

Given the following user header array:

```
0094USGS GAGE ID:012345678; LATITUDE:412345; LONGITUDE:1212345;
DATUM:1234.56; OPERATOR:JOHN DOE;
```

```
C      Example 1
C      Search for the latitude and longitude stored in the user
C      header array, getting one item at a time.
C
       REAL HEADU(NHEADU)
       CHARACTER CLABEL*20, CITEM*20
C
       CALL ZUSTFH ('LATITUDE', CITEM, 1, IPOS, HEADU, NHEADU,
ISTAT)
       IF (ISTAT.NE.0) GO TO 900
       WRITE (6,*)'The latitude is: ', CITEM
```

```
      CALL ZUSTFH ('LONGITUDE', CITEM, 1, IPOS, HEADU, NHEADU,
ISTAT)
      IF (ISTAT.NE.0) GO TO 900
      WRITE (6,*)'The longitude is: ', CITEM


C     Example 2
C     Search for the latitude and longitude stored in the user
C     header array, getting both items at the same time.
C
      REAL HEADU(NHEADU)
      CHARACTER CLABEL(2)*20, CITEM(2)*20
C
      CLABEL(1) = 'LATITUDE'
      CLABEL(2) = 'LONGITUDE'
      CALL ZUSTFH (CLABEL, CITEM, 2, IPOS, HEADU, NHEADU, ISTAT)
      IF (ISTAT.NE.0) GO TO 900
      WRITE (6,*)'The latitude is: ', CITEM(1)
      WRITE (6,*)'The longitude is: ', CITEM(2)
```

Examples 1 and 2 would print:

```
The latitude is: 412345
The longitude is: 1212345
```

```
C     Example 3
C     Get all labels and items from the user header.
C
      REAL HEADU(NHEADU)
      CHARACTER CLABEL*20, CITEM*20
C
      IPOS = 0
      NITEM = 0
 20   CONTINUE
      CALL ZUSTFH (CLABEL, CITEM, NITEM, IPOS, HEADU, NHEADU,
ISTAT)
      IF (ISTAT.NE.0) GO TO 900
      WRITE (6,40) CLABEL(1), CITEM(1)
 40   FORMAT (' Label: ',A,',  Item: ',A)
      IF (IPOS.GT.0) GO TO 20
```

Using the same header array, this example would print:

```
Label: USGS GAGE ID   ,   Item: 012345678
Label: LATITUDE       ,   Item: 412345
Label: LONGITUDE      ,   Item: 1212345
Label: DATUM          ,   Item: 1234.56
Label: OPERATOR       ,   Item: JOHN DOE
```

9.3 ZCHECK - Check if a Record Exists

Purpose:

ZCHECK checks for the presence of a record in a DSS file. If the record exists, the number of elements in the user header and data arrays are returned.

Calling Sequence:

```
CALL ZCHECK (IFLTAB, CPATH, NPATH, NHEAD, NDATA, LFOUND)
```

Declarations:

```
INTEGER IFLTAB(600), NPATH, NHEAD, NDATA
CHARACTER CPATH*80
LOGICAL LFOUND
```

Argument Description:

IFLTAB	Input/ Output	The DSS work space used to manage the DSS file. This is the same array used in the ZOPEN call.
CPATH	Input	The pathname of the record to check. The pathname does not have to follow the standard conventions.
NPATH	Input	The number of characters in CPATH. Trailing blanks must be excluded.
NHEAD	Output	The number of real elements stored in the user header array.
NDATA	Output	The number of real elements stored in the data array.
KDATA	Input	The dimension of the array DATA. No more than KDATA values will be returned at a time.
NDATA	Output	The number of elements returned in the data array.
LFOUND	Output	A logical status variable that is returned .TRUE. if the record was found, or .FALSE. if it does not exist.

Remarks:

ZCHECK is generally used only by lower level routines. Programs typically call ZDTYPE instead of ZCHECK. No messages are printed by ZCHECK.

Note: Earlier versions of DSS returned the lengths NHEAD and NDATA in short integer words. Version 6 now returns those values for real arrays. This change is the only incompatibility between DSS version 6 and earlier versions.

9.4 ZRECIN - Display Information About a Record

Purpose:

ZRECIN displays information about a record. This information is the same as that printed in the CHECK command for DSSUTL. It contains the type of data, its last written date and time, and the program that wrote it. If it is time-series data, then any data compression statistics or data flag information is also displayed. Examples of the output are given below.

Calling Sequence:

```
CALL ZRECIN (IFLTAB, IUNIT, MLEVEL, CPATH, BUFF, KBUFF, LFOUND)
```

Declarations:

```
INTEGER IFLTAB(600), IUNIT, MLEVEL, KBUFF
REAL BUFF(KBUFF)
CHARACTER CPATH*80
LOGICAL LFOUND
```

Argument Description:

IFLTAB	Input/ Output	The DSS work space used to manage the DSS file. This is the same array used in the ZOPEN call.
IUNIT	Input	The unit number to write the information to. This is usually connected to the standard output.
MLEVEL	Input	The message level. If this is set to zero, only a statement indicating if the record exists or not is displayed. If MLEVEL is two or more, information about the record is displayed.
CPATH	Input	The record pathname.
BUFF	Input	A temporary buffer array. The record header and compression header are read into this array. Its size should be a minimum of seventy elements.
KBUFF	Input	The dimension of BUFF.
LFOUND	Output	A logical status variable that is returned .TRUE. if the record was found, or .FALSE. if it does not exist.

Remarks:

ZRECIN is designed to display information, not as a means of a program obtaining information about a record. A program should not attempt to read information from it, as items may be moved in the display in different versions of DSS.

Example Displays:

```
Record Found:
/ALLEGHENY/NATP/PRECIP-INC/01JUL1989/1HOUR/OBS/
Regular-interval time series;  Tag: T1284
Last Written on 11OCT89,  at 09:58  by Program:  Undefi
Version:   1;  Number of data:  744;  Space Allocated:   10
Compressed to  4.3%
Compression Method:  3;  Repeat + Delta
Precision: -2;  Element Size: 1;  Base:        0.00;  User set base: F

Record Found:
/ALLEGHENY/BRFP/FLOW/01APR1990/1HOUR/REV/
Regular-interval time series;  Tag: T299
Last Written on 10JUL90,  at 16:43  by Program:  DATCHK
Version:   2;  Number of data:  720;  Space Allocated: 1440
Data flags set.
```

9.5 ZFILST - Display Information About a DSS File

Purpose:

ZFILST displays status information about a DSS file. The information displayed is the same as that displayed in the DSSUTL FQ (File Query) command. It includes items such as the file size, amount inactive space, and file statistics. The information is written to unit MUNIT (which can be reset from the default by subroutine ZSET).

Calling Sequence:

```
CALL ZFILST (IFLTAB)
```

Declarations:

```
INTEGER IFLTAB(600)
```

Argument Description:

IFLTAB	Input/ Output	The DSS workspace used to manage the DSS file. This is the same array used in the ZOPEN call.

Remarks:

ZFILST is designed to display information, not as a means of a program obtaining information about the file. A program should not attempt to read information from it, as items may be moved in the display in different versions of DSS.

Example Displays

```
-------------------------------------------------------------------------------
DSS File MASTDB;   Created on 10JAN90;   DSS Version 6-EA
Number of Records:  3840;       Pointer Utilization:  2.50
File Size:  12210.4 Kilobytes;  Percent Inactive Space:  6.06
Max Hash Code: 1024;            Stable Hash Table
Numb Bins per Block:  32;       Size of Bin:  112 words
Bins Used:    619;          Overflow Bins:    181
Hash Codes Used:   438;         Max Paths for one Hash Code:   11
Average Number of Paths to search:   8.8;   (Max Hash Code:  262)
-------------------------------------------------------------------------------
```

9.6 ZCOREC - Copy a Record

Purpose:

ZCOREC copies a record from one DSS file to another, or duplicates a record within the same file. A regular-interval time series record can also be compressed (or un-compressed) according to the new file's default data compression setting, if desired.

Calling Sequence:

```
      CALL ZCOREC (IFTOLD, IFTNEW, CPOLD, CPNEW, BUFF1, KBUFF1,
     *  BUFF2, KBUFF2, ISTAT)
```

Declarations:

```
      INTEGER IFTOLD(600), IFTNEW(600), KBUFF1, KBUFF2, ISTAT
      REAL BUFF1(KBUFF1), BUFF2(KBUFF2)
      CHARACTER CPOLD*80, CPNEW*80
```

Argument Description:

IFTOLD	Input/ Output	The DSS work space used to manage the DSS file. This is the IFLTAB for the DSS file to copy **from**.
IFTNEW	Input/ Output	The DSS work space used to manage the DSS file. This is the IFLTAB for the DSS file to copy **to**. If the record is to be duplicated within the same file, this should be array IFTOLD.
CPOLD	Input	The pathname of the record to copy.
CPNEW	Input	The pathname that the copied record is to have. If the record is to be copied from one file to another, then this may be CPOLD. If the record is being duplicated within the file, then this pathname cannot be the same as CPOLD.
BUFF1	Input	A scratch array that will temporarily hold the data, or portions of the data. If the data is time series and is to be re-compressed, this must be large enough to hold all the data within the record. Otherwise BUFF1 can be smaller, as buffered reads and writes are used. However, it is more efficient to make this array large enough to hold all of the data within the record. A typical dimension for this array is 750.
KBUFF1	Input	The dimension of BUFF1.

BUFF2	Input	A scratch array that will temporarily hold the internal header array. This array does not need to be as large as BUFF1. A typical dimension for this array is one hundred.
KBUFF2	Input	The dimension of BUFF2.
ISTAT	Output	A status parameter indicating the success of the operation. If ISTAT is returned as zero, then the record was copied successfully. If ISTAT is other than zero, the record was not copied. The possible values are:

ISTAT	**Description**
0	Successful operation.
1	The record to be copied (CPOLD) does not exist.
2	The new record (CPNEW) already exists, and write protection was set for the file.
-1	KBUFF1 or KBUFF2 is zero.
-2	The buffers supplied (BUFF1 and BUFF2) are too small for this record. The size required will be printed if the message level is two or greater.
-12	The file being copied to is in a read access only mode.

Remarks:

Except for time series data that is to be re-compressed, the data is copied by buffered reads and writes. Thus only portions of the data and user header arrays will be copied at a time (up to KBUFF1 values). If space is available, setting KBUFF1 to the size of the data array is most efficient.

Time series data can be re-compressed (or un-compressed) using the default data compression settings of the DSS file being copied to by calling ZSET with a parameter of "COMP", prior to ZCOREC. This will remain set until explicitly set "OFF" by a subsequent call to ZSET.

Example 1:

```
C     Copy a record from one DSS file to another.
      INTEGER IFTOLD(600), IFTNEW(600)
      CHARACTER CPATH*80
      PARAMETER (KBUFF1=750, KBUFF2=100)
      REAL BUFF1(KBUFF1), BUFF2(KBUFF2)
      CHARACTER CNOLD*64, CNNEW*64
C
      READ (5,*) CNOLD, CNNEW
      CALL ZOPEN (IFTOLD, CNOLD, ISTAT)
```

```
        IF (ISTAT.NE.0) GO TO 900
        CALL ZOPEN (IFTOLD, CNNEW, ISTAT)
        IF (ISTAT.NE.0) GO TO 900
  C
        CPATH = '/SACRAMENTO/I ST/FLOW/01JAN1980/1HOUR/OBS/'
  C
        CALL ZCOREC (IFTOLD, IFTNEW, CPATH, CPATH, BUFF1, KBUFF1,
       *  BUFF2, KBUFF2, ISTAT)
        IF (ISTAT.NE.0) GO TO 910
  C
```

Example 2:

```
  C     Duplicate a record within a DSS file.
        INTEGER IFLTAB(600)
        CHARACTER CPOLD*80, CPNEW*80
        PARAMETER (KBUFF1=750, KBUFF2=100)
        REAL BUFF1(KBUFF1), BUFF2(KBUFF2)
        CHARACTER CNAME*64
  C
        READ (5,*) CNAME
        CALL ZOPEN (IFLTAB, CNAME, ISTAT)
        IF (ISTAT.NE.0) GO TO 900
  C
        CPOLD = '/SACRAMENTO/I ST/ELEV-DAMAGE//1980/PLAN B/'
        CPNEW = '/SACRAMENTO/I ST/ELEV-DAMAGE//1980/REVISED/'
  C
        CALL ZCOREC (IFLTAB, IFLTAB, CPOLD, CPNEW, BUFF1, KBUFF1,
       *  BUFF2, KBUFF2, ISTAT)
        IF (ISTAT.NE.0) GO TO 910
  C
```

9.7 ZCOFIL - Copy a DSS File

Purpose:

ZCOFIL copies a DSS file to a new DSS file, or appends the file to an existing DSS file. ZCOFIL copies only valid data, so any inactive space is not copied. ZCOFIL is called by DSSUTL to "squeeze" a DSS file. Regular interval time series records can also be compressed (or un-compressed) according to the new file default data compression setting, if desired. ZCOREC should be called if less than the entire file is to be copied.

Calling Sequence:

```
    CALL ZCOFIL (IFTOLD, IFTNEW, BUFF1, KBUFF1, BUFF2, KBUFF2,
*   LUNDEL, LRETAG)
```

Declarations:

```
    INTEGER IFTOLD(600), IFTNEW(600), KBUFF1, KBUFF2
    REAL BUFF1(KBUFF1), BUFF2(KBUFF2
    LOGICAL LUNDEL, LRETAG
```

Argument Description:

IFTOLD	Input/ Output	The DSS work space used to manage the DSS file. This is the IFLTAB for the DSS file to copy **from**.
IFTNEW	Input/ Output	The DSS work space used to manage the DSS file. This is the IFLTAB for the DSS file to copy **to**.
BUFF1	Input	A scratch array that will temporarily hold data and internal arrays. The minimum dimension of this array should be 750 elements.
KBUFF1	Input	The dimension of BUFF1.
BUFF2	Input	A scratch array that will temporarily hold data and internal arrays. The minimum dimension of this array should be 750 elements.
KBUFF2	Input	The dimension of BUFF2.
LUNDEL	Input	A logical flag that should be set to .TRUE. if records that have been deleted, but not yet physically removed, should be copied. (They will be undeleted in the new file).
LRETAG	Input	A logical flag that should be set to .TRUE. if the records should be assigned new tag identifiers when they are copied.

The tags assigned will use the file tag settings. (See Chapter 7 for information on tags.)

Remarks:

ZCOFIL reads through the file in a "brute force" fashion. Thus, if a file somehow becomes damaged, ZCOFIL will copy all data that is recoverable. (It cannot tell if data itself has become corrupt.)

Buffered reads and writes are used for larger data records. The buffer arrays passed do not necessarily need to be as large as the record sizes.

9.8 ZRENAM - Rename a Record

Purpose:

ZRENAM changes the pathname of a record in a DSS file.

Calling Sequence:

```
CALL ZRENAM (IFLTAB, CPATHO, NPATHO, CPATHN, NPATHN, LFOUND)
```

Declarations:

```
INTEGER IFLTAB(600), NPATHO, NPATHN
CHARACTER CPATHO*80, CPATHN*80
LOGICAL LFOUND
```

Argument Description:

IFLTAB	Input/ Output	The DSS work space used to manage the DSS file. This is the same array used in the ZOPEN call.
CPATHO	Input	The pathname of the record to be renamed.
NPATHO	Input	The number of characters in CPATHO.
CPATHN	Input	The new pathname to be given to the record.
NPATHN	Input	The number of characters in CPATHN
LFOUND	Output	A logical status variable that is returned .TRUE. if the record was found and renamed, or .FALSE. if the original record did not exist in the DSS file.

Remarks:

ZRENAM does not change the record's tag or any other information. With a message level of three or greater, ZRENAM prints a message with the old pathname and the new pathname. Error messages are printed with a message level of two or greater.

9.9 ZDELET - Delete a Record

Purpose:

ZDELET deletes a record from a DSS file by flagging a record status cell. The data is not physically removed until the file is squeezed by DSSUTL. A deleted record can be undeleted by DSSUTL or the subroutines ZUNDEL and ZUDALL (until the file is squeezed).

Calling Sequence:

```
CALL ZDELET (IFLTAB, CPATH, NPATH, LFOUND)
```

Declarations:

```
INTEGER IFLTAB(600), NPATH
CHARACTER CPATH*80
LOGICAL LFOUND
```

Argument Description:

IFLTAB	Input/ Output	The DSS work space used to manage the DSS file. This is the same array used in the ZOPEN call.
CPATH	Input	The pathname of the record to eliminate. This pathname does not have to follow the standard conventions.
NPATH	Input	The number of characters in CPATH. Trailing blanks must be excluded.
LFOUND	Output	A logical status variable that is returned .TRUE. if the record was deleted, or .FALSE. if the record did not exist in the DSS file.

Remarks:

ZDELET is usually only called by utility programs. With a message level of three or greater, ZDELET prints a message indicating the record was deleted. Error messages are printed with a message level of two or greater.

9.10 ZUNDEL - Undelete a Record

Purpose:

ZUNDEL recovers a DSS record that was previously deleted by ZDELET by modifying a record status cell. The DSS file must not have been squeezed by DSSUTL since the record was deleted (as that will physically remove the record). All deleted records within a file can be recovered by subroutine ZUDALL. ZUDALL can also determine what records are available to recover.

Calling Sequence:

CALL ZUNDEL (IFLTAB, CPATH, NPATH, ISTAT)

Declarations:

INTEGER IFLTAB(600), NPATH, ISTAT
CHARACTER CPATH*80

Argument Description:

IFLTAB	Input/ Output	The DSS work space used to manage the DSS file. This is the same array used in the ZOPEN call.
CPATH	Input	The pathname of the record to recover.
NPATH	Input	The number of characters in CPATH. Trailing blanks must be excluded.
ISTAT	Output	A status parameter indicating the success of the operation. If ISTAT is returned with zero, then the record was successfully recovered. If ISTAT is non-zero, an error occurred. The possible values are:

ISTAT	Description
0	Successful operation.
1	The record could not be found.
2	The record already exists (it was not deleted).

Remarks:

With a message level of three or greater, ZUNDEL prints a message indicating the record was recovered. Error messages are printed with a message level of two or greater.

9.11 ZUDALL - Undelete All Records in a DSS File

Purpose:

ZUDALL recovers all records in a DSS file that were previously deleted by ZDELET. The DSS file must not have been squeezed by DSSUTL since the records were deleted (as that will physically remove the records). ZUDALL can also display a list of the deleted records within the file that are available to recover.

Calling Sequence:

CALL ZUDALL (IFLTAB, IUNIT)

Declarations:

INTEGER IFLTAB(600), IUNIT

Argument Description:

IFLTAB	Input/ Output	The DSS work space used to manage the DSS file. This is the same array used in the ZOPEN call.
IUNIT	Input	If the records available to recover are to be displayed, this is the unit number to write the pathnames of those records to. If the records are to be recovered, IUNIT must be zero.

Remarks:

To only display the deleted records in a file, set IUNIT to the unit number of the output (otherwise IUNIT must be zero). ZUDALL will not recover any records when displaying available records to recover. The following message is printed for each record available to recover:

```
---ZUNDEL;  Available:  pathname
```

IUNIT must be set to zero to actually recover records.

With a message level of 3 or greater, ZUDALL prints a message with the pathname for each record that was recovered. If no records are available to recover, ZUDALL prints a message indicating so

9.12 ZDEBUG - Display Coded Information from the File or the IFLTAB Array

Purpose:

ZDEBUG displays coded information in the IFLTAB array or an array containing information read from the DSS file. ZDEBUG is used internally in debugging possible damaged areas in a file, and for installing the DSS software on a new computer. It is not intended for debugging programs accessing DSS.

Calling Sequence:

```
CALL ZDEBUG (MUNIT, IARRAY, IADD, ILEN)
```

Declarations:

```
INTEGER MUNIT, IARRAY(*), IADD, ILEN
```

On MS-DOS microcomputers, the address must be INTEGER*4: INTEGER*4 IADD
On Harris computers, the address must be INTEGER*6: INTEGER*6 IADD

Argument Description:

MUNIT	Input	The unit number to print out the information to.
IARRAY	Input	The DSS IFLTAB array or other array to display.
IADD	Input	The file address or location within the array. (For example, if you were displaying the IFLTAB array from word thirty-two to fifty, this would be the number thirty-two.) The address is displayed in the left-hand column, and incremented for each word. It is used for informational purposes only.
ILEN	Input	The number of large integer words to print.

Remarks:

The output from ZDEBUG occupies 132 columns. Each word in the array is printed as a large integer, a character string, a real number, two small integers, and four or six bytes (depending on the computer). The address, and the address' record and word are printed on the left side of the information.

Example

If ZDEBUG is called with the IFLTAB array and a length of thirty after a call to ZOPEN, the output might appear as the following:

```
Address  Rec  Word  Offset        Large Int     Char          Real    Small     Ints    Bytes
      1    1     1     (1)                6   ~~~~~~          0.00        0        6     0    0    0    0     0    6
      2    1     2     (2)            13579   ~~~~5~          0.00        0   135790     0    0    0          53   11
      3    1     3     (3)               71   ~~~~~G          0.00        0       71     0    0    0    0     0   71
      4    1     4     (4)               71   ~~~~~G          0.00        0       71     0    0    0    0     0   71
      5    1     5     (5)                1   ~~~~~~          0.00        0        1     0    0    0    0     0    1
      6    1     6     (6)                1   ~~~~~~          0.00        0        1     0    0    0    0     0    1
      7    1     7     (7)                0   ~~~~~~          0.00        0        0     0    0    0    0          0
      8    1     8     (8)             4929   ~~~~~A  ************        0     4929     0    0    0    0    19   65
      9    1     9     (9)          2207838   ~~~!~^  ************        0  2207838     0    0    0   33   176   94
     10    1    10    (10)                1   ~~~~~~          0.00        0        1     0    0    0    0     0    1
     11    1    11    (11)                0   ~~~~~~          0.00        0        0     0    0    0    0     0    0
     12    1    12    (12)                0   ~~~~~~          0.00        0        0     0    0    0    0     0    0
     13    1    13    (13)                0   ~~~~~~          0.00        0        0     0    0    0    0     0    0
     14    1    14    (14)                0   ~~~~~~          0.00        0        0     0    0    0    0     0    0
     15    1    15    (15)   49624753831936   ZDSS~~          0.71  5915731  5439488    90   68   83   83     0    0
     16    1    16    (16)             4193   ~~~~~a  ************        0     4193     0    0    0    0    16   97
     17    1    17    (17)             4194   ~~~~~b  ************        0     4194     0    0    0    0    16   98
     18    1    18    (18)                0   ~~~~~~          0.00        0        0     0    0    0    0     0    0
     19    1    19    (19)   29784033787904   6-EA~~          0.42  3550533  4259840    54   45   69   65     0    0
     20    1    20    (20)          2207838   ~~~!~^  ************        0  2207838     0    0    0   33   176   94
     21    1    21    (21)           127762   ~~~~~~          0.00        0   127762     0    0    0    1   243   18
     22    1    22    (22)   27041739132473    10JAN  ************  3223626  4279865    49   48   74   65    78   57
     23    1    23    (23)   26388279066624   0~~~~~          0.37  3145728        0    48    0    0    0     0    0
     24    1    24    (24)   26496278285881   02JAN9  ************  3158602  4279865    48   50   74   65    78   57
     25    1    25    (25)   26938034880512   1~~~~~          0.38  3211264        0    49    0    0    0     0    0
     26    1    26    (26)   26511175398202   09:13:  ************  3160378  3224378    48   57   58   49    51   58
     27    1    27    (27)   27049704030208   14~~~~          0.38  3224576        0    49   52    0    0     0    0
     28    1    28    (28)          1383873   ~~~~~~          0.00        0  1383873     0    0    0   21    29  193
     29    1    29    (29)             1024   ~~~~~~          0.00        0     1024     0    0    0    0     4    0
     30    1    30    (30)                2   ~~~~~~          0.00        0        2     0    0    0    0     0    2
```

10 Data Compression Subroutines

Regular-interval time series data may be compressed by one or more of three methods. The method may be selected by the user or by the storing program based on the kind of data that is being stored. At this time, no other data types can be compressed by DSS.

The first method is a repeat counter scheme that flags duplicate values. It uses one bit per value to indicate a repeated value. It is often used for precipitation data, and can compress some precipitation records by up to 97 percent. This compression method should never be used for data that is updated frequently (e.g., entering real time data in a master database file), as it would cause the record to expand often and require excessive rewrites.

The second method (called the delta method) compresses data by storing the differences between the data values and the minimum value in the record. This is designed for data where the difference between the maximum value and the minimum value is not too large, and the "precision" of the values (the number of digits to the right of the decimal place) is known. The software determines the amount of space required for the data based on the difference between the maximum and the minimum (or base) value (excluding missing data flags), and the precision number. If data is to be updated frequently with this method (e.g., entering real-time data in a master database file), the base value and a storage size can be specified. This allows the software to update the data without having to re-compute a base value (and possibly re-compress the record) each time. Data compressed by this method are typically precipitation and stage values, and are compressed by fifty or seventy-five percent of their original size.

The third type of compression stores three significant digits for each value, and is often used to compress flow data. Data records compressed under with method are reduced in size by fifty percent.

The repeat method can be used in combination with the differences method or significant digits method. The differences method may not be used with the significant digits method.

The methods discussed in this chapter are often referenced by the software as numbers. The numbers and their corresponding methods are as follows:

Number	Method
0	NONE
1	REPEAT
2	DELTA
3	REPEAT + DELTA
4	SIGNIFICANT DIGITS
5	REPEAT + SIGNIFICANT DIGITS

The delta method requires a precision exponent parameter indicating the accuracy of the data. If the data to be stored is measured to the nearest hundredth (0.01) (e.g., precipitation), the precision exponent would be negative two. If the data is to the nearest thousandth, the precision exponent would be negative three.

In addition, a "base value" and "data size" parameter may be specified for the delta method. These parameters are typically only used with "real-time data", data that is updated frequently, and only to increase the efficiency of storing future data. The base value is the expected minimum value that the data will obtain for that record. For example, the base value for incremental precipitation would be 0.0. The data size parameter indicates whether one or two bytes should be pre-allocated for each data value. One byte allocates a difference of 256 units, two bytes allocates a difference of 65,536 units. Typically, hourly precipitation would pre-allocate only one byte (up to 2.56 inches per hour), whereas reservoir elevations would pre-allocate two bytes (up to 65.536 feet difference). If the data changes so that either of the selected values is invalid, the software will automatically select new values and re-compress the data. If the parameters are not specified, the software will automatically select values based upon the data.

There are two ways to instruct the DSS software to compress regular-interval time series data. One is to set data compression methods for the entire DSS file based on matching pathname parts, and the other is to specify the compression method as an argument in the subroutine that stores the data.

In the first procedure a header section in the file is set with pathname part(s) and compression methods. If a new record is stored, and its pathname part(s) match those in the file header section, then that record will be compressed with the method specified. For example, one may designate that records with a pathname C part that begins with "PRECIP" be compressed with the REPEAT + DELTA method, and a pathname C part of "FLOW" be compressed with the SIGNIFICANT DIGITS method. As many parts/methods as desired may be defined in the file header. Only new records are compressed by this means, unless the "C" option is used in the DSSUTL squeeze command. The data compression file header may be set by the Data Compression command in DSSUTL, or by calling the subroutine ZSETCI.

For the second procedure subroutine ZSCOMP is called prior to ZSRTS to define a data compression method, or subroutine ZSRTSX is called instead of ZSRTS, with the data compression method passed as an argument. In both cases the method passed will override any default file methods set.

The compression method used, and its associated parameters, may be obtained for a record by calling subroutine ZDCINF after retrieving the data. Compression information about a record can be printed with subroutine ZRECIN, as described in Chapter 9. A DSS file's default compression methods can be printed with subroutine ZPRTCI.

Data compression may not be used in conjunction with data flags.

10.1 ZSCOMP - Set Data Compression for a Record

Purpose:

ZSCOMP sets data compression parameters when storing data with subroutine ZSRTS. ZSCOMP must be call just prior to ZSRTS, and the compression parameters apply only to that call to ZSRTS.

Calling Sequence:

```
CALL ZSCOMP (ICOMP, BASEV, LBASEV, LHIGH, IPREC)
```

Declarations:

```
INTEGER ICOMP, IPREC
REAL BASEV
LOGICAL LBASEV, LHIGH
```

Argument Description:

ICOMP	Input	The data compression method to use, as described in the introduction to this chapter.
BASEV	Input	When the delta data compression method is used, the base value may be specified by setting this argument to the base value and LBASEV to .TRUE.. If the delta method is not used, this argument is ignored.
LBASEV	Input	A logical flag indicating if the argument BASEV has been set. To let the compression software select a base value, set this argument to .FALSE.. If the delta method is not used, this argument is ignored.
LHIGH	Input	When the delta data compression method is used, setting LHIGH to .TRUE. will pre-allocate two bytes of storage per data value. If LHIGH is set to .FALSE., the compression software will select the storage size based on the data. If the delta method is not used, this argument is ignored.
IPREC	Input	When the delta data compression method is used; this defines the precision exponent of the data (required). The precision exponent may range from negative six to positive six.

Remarks:

ZSCOMP is normally used when adding data compression capabilities to existing code. The subroutine ZSRTSX includes data compression parameters as arguments, and is typically called in place of ZSCOMP.

The compression parameters passed in for ZSCOMP override any default file compression methods set (unless ICOMP is set to zero). To disallow compression for this data, set ICOMP to negative one.

ZSCOMP must be called prior to each call to ZSRTS to set data compression parameters. The parameters set by ZSCOMP are reset upon exit from ZSRTS. If the record is already compressed, it will be re-compressed with the new method specified.

Example:

```
C      If precipitation or flow data is to be stored, compress the data.
C      For precipitation, use the delta + repeat method (method #3).
C      For flow data, use the significant digits method (method #4).
C
       . . .
C
       CALL ZPATH (CA, CB, CC, CD, CE, CF, CPATH, NPATH)
C
C      Is this precip data?  (If so, compress with delta + repeat).
       IF (CC(1:6).EQ.'PRECIP') THEN
C      Set the compression method to 3, with a base value of 0.0.
C      Store data to the nearest hundredth of an inch.  Do not
C      require a two byte space pre-allocation.
       CALL ZSCOMP (3, 0.0, .TRUE., .FALSE., -2)
       ENDIF
C
C      Is this flow data?  (If so, compress with significant digits).
       IF (CC(1:4).EQ.'FLOW') THEN
C      Set the compression method to 4.  All other arguments
C      are ignored.
       CALL ZSCOMP (4, DUM, LDUM, LDUM, IDUM)
       ENDIF
C
C      Now store the data.
       CALL ZSRTS (IFLTAB, CPATH, CDATE, CTIME, NVALS,
     * VALUES, CUNITS, CTYPE, IPLAN, ISTAT)
```

10.2 ZDCINF - Get Data Compression Information for a Record

Purpose:

ZDCINF returns data compression parameters for the last regular-interval time series record read. This includes the compression method, and if the delta method was used, the base value, the precision, and the number of bytes allocated for each value. The record must have been retrieved to use this subroutine. To obtain data compression information about a record that has not been read, call subroutine ZRECIN.

Calling Sequence:

```
CALL ZDCINF (ICOMP, BASEV, LBASEV, ISIZE, IPREC, ISTAT)
```

Declarations:

```
INTEGER ICOMP, IPREC, ISIZE, ISTAT
REAL BASEV
LOGICAL LBASEV
```

Argument Description:

ICOMP	Output	The data compression method used, as described in the introduction to this chapter.
BASEV	Output	The base (minimum) value of data when the delta compression method is used. If the delta method is not used, this argument is undefined.
LBASEV	Output	A logical flag set to .TRUE. if the base value was set for the delta method. If the delta method is not used, this argument is undefined.
ISIZE	Output	If the delta compression method was used, this argument will contain the number of bytes (one or two) allocated for each data value. If the delta method is not used, this argument is undefined.
IPREC	Output	If the delta compression method was used, this argument will contain the precision exponent. If the delta method is not used, this argument is undefined.
ISTAT	Output	A status parameter set to zero if the data was compressed. If ISTAT is non-zero, the data was not compressed and the subroutine arguments are undefined.

Remarks:

ZDCINF can only be called after the data has been retrieved with ZRRTS or ZRRTSX. If the data was not compressed, ISTAT will be returned negative, and the arguments will be unchanged.

Example:

```
C       Retrieve time-series data and print the data compression
C       information about it.
C
C       Retrieve the data
        CALL ZRRTS (IFLTAB, CPATH, CDATE, CTIME, NVALS,
     *  VALUES, CUNITS, IOFSET, ISTAT)
C
C       "Fatal" error?
        IF (ISTAT.GE.10) GO TO 950
C       No data?
        IF (ISTAT.GE.4) GO TO 960
C
C       Write pathname, units.
        CALL CHRLNB (CPATH, NPATH)
        WRITE (6,40) CPATH(1:NPATH), CUNITS, CTYPE
 40     FORMAT (' Pathname: ',A,/,' Units: ',A,T20,'Type: ',A)
C
C       Get compression information.
        CALL ZDCINF (ICOMP, BASEV, LBASEV, ISIZE, IPREC, JSTAT)
C
        IF (JSTAT.EQ.0) THEN
C         Does this compression include the delta method?
          IF ((ICOMP.EQ.2).OR.(ICOMP.EQ.3)) THEN
            WRITE (6,60) ICOMP, LBASEV, BASEV, ISIZE, IPREC
 60         FORMAT (' Compression Method:',I3,'  Based Set:',L2,
     *      ' Base: ',F6.1,/,' Size Allocated:',I2,'  Precision:',I3)
          ELSE
            WRITE (6,80) ICOMP
 80         FORMAT (' Compression Method:',I3)
          ENDIF
        ELSE
          WRITE (6,100)
 100      FORMAT (' No Compression Used')
        ENDIF
C
```

10.3 ZSETCI - Set Default Data Compression for a DSS File

Purpose:

ZSETCI sets the data compression header for a DSS file. The compression method set will be used on all new regular-interval time series records that match the defined pathname parts, unless a compression method is specified when storing the data with ZSRTSX or ZSCOMP. For example, one may designate that records with a pathname C part that begins with "PRECIP" be compressed with the REPEAT + DELTA method. Any time a new record is written (from any program), and the C part begins with "PRECIP", the data will be compressed automatically with the REPEAT + DELTA method (unless overridden). As many parts/methods settings as desired may be defined in the file header. Only new records are compressed by this means, unless the "C" option is used in the DSSUTL squeeze command. File compression settings may also be removed with ZSETCI.

Calling Sequence:

```
      CALL ZSETCI (IFLTAB, CPARTS, LPARTS, ICOMP, BASEV, LBASEV,
     *  LHIGH, IPREC, ISTAT)
```

Declarations:

```
      INTEGER IFLTAB(600), ICOMP, IPREC, ISTAT
      CHARACTER CPARTS(6)*32
      REAL BASEV
      LOGICAL LBASEV, LHIGH, LPARTS(6)
```

Argument Description:

IFLTAB	Input/ Output	The DSS work space used to manage the DSS file. This is the same array used in the ZOPEN call.
CPARTS	Input	A six element character array containing the pathname parts to match. Only those elements that are to be matched (as indicated by LPARTS) need to be defined. The first element of CPARTS corresponds to the A part of the pathname, the second element to the B part, etc. The "@" character may be used as a wild character at the end (only) of a portion of a part, so only the beginning of the part is matched. For example, if CPARTS(3) = 'PRECIP@', "PRECIP-CUM", and "PRECIP- ", will both match.
LPARTS	Input	A six element logical array indicating which parts are to be matched. The first element of LPARTS corresponds to CPARTS(1) (the "A" part), the second element to CPARTS(2), etc.. If an element of LPARTS is .FALSE., that part will not

		be matched. If an element is .TRUE., then that part must match.
ICOMP	Output	The data compression method to use, as described in the introduction to this chapter. To delete a file setting, set ICOMP to zero.
BASEV	Output	When the delta data compression method is used, the base value may be specified by setting this argument to the base value and LBASEV to .TRUE.. If the delta method is not used, this argument is ignored.
LBASEV	Input	A logical flag that should be set to .TRUE. if the argument BASEV has been set.
LHIGH	Output	When the delta data compression method is used, setting LHIGH to .TRUE. will pre-allocate two bytes of storage per data value. If LHIGH is set to .FALSE., the compression software will select the storage size based on the data.
IPREC	Output	When the delta data compression method is used; this defines the precision exponent of the data (required). The precision exponent may range from negative six to positive six.
ISTAT	Output	A status parameter indicating the success of the operation. If ISTAT is returned with zero, then the file header was changed successfully. The possible values are:

ISTAT	**Description**
0	The file header was successfully modified.
-1	The part identifiers were not found (no match). This only applies when deleting a file setting (ICOMP is zero).
1	An invalid compression method (ICOMP) was specified. ICOMP may range from 0 to 5
2	An invalid precision value (IPREC) was specified for use with the delta compression method. IPREC may range from negative six to positive six.

Remarks:

If the parts match an existing file setting, then that compression method will be replaced. A file setting can be deleted by making ICOMP zero (with matching CPARTS).

As many compression sets may be defined for a DSS file as desired. If a pathname matches more than one set, the first matching set encountered is used.

The data compression header may be displayed with subroutine ZSETCI.

Example:

```
C      Set a dss file's data compression header so that precipitation
C      and flow data are automatically compressed.
C
       CHARACTER CPARTS(6)*32
       LOGICAL LPARTS(6)
       . . .
C
C      Attach units, open the DSS file, etc.
       CALL ZOPEN (...
C
C      Use the repeated values + delta method for precipitation data.
C      Set the base to zero, and the precision to a hundredth (0.01).
       DO 20 I=1,6
         LPARTS(I) = .FALSE.
 20    CONTINUE
       CPARTS(3) = 'PRECIP@'
       LPARTS(3) = .TRUE.
       CALL ZSETCI (IFLTAB, CPARTS, LPARTS, 3, .TRUE., 0.0,
     * FALSE., -2, ISTAT)
       IF (ISTAT.NE.0) GO TO 900
C
C      Use the significant digits method for flow data.
       DO 40 I=1,6
         LPARTS(I) = .FALSE.
 40    CONTINUE
       CPARTS(3) = 'FLOW@'
       LPARTS(3) = .TRUE.
       CALL ZSETCI (IFLTAB, CPARTS, LPARTS, 4, .FALSE., 0.0,
     * .FALSE., IDUM, ISTAT)
       IF (ISTAT.NE.0) GO TO 900
C
```

10.4 ZPRTCI - Print the Default Data Compression for a DSS File

Purpose:

ZPRTCI prints information about a DSS file's data compression header. All compression information, or a portion of it based on matching pathname parts, can be printed. The information is written to the DSS message unit (MUNIT).

Calling Sequence:

```
CALL ZPRTCI (IFLTAB, LALL, CPARTS)
```

Declarations:

```
INTEGER IFLTAB(600)
CHARACTER CPARTS(6)*32
LOGICAL LALL
```

Argument Description:

IFLTAB	Input/ Output	The DSS work space used to manage the DSS file. This is the same array used in the ZOPEN call.
LALL	Input	A logical variable indicating if all information is to be printed out, or just information for specific parts. If LALL is .TRUE., then all information is printed, and the argument CPARTS is ignored. If LALL is .FALSE., the CPARTS is examined for matching parts to determine what compression information should be printed.
CPARTS	Input	A six element character array containing the pathname parts to match when LALL is .TRUE.. The first element of CPARTS corresponds to the A part of the pathname, the second element to the B part, etc. If a part is not to be matched, it must be blank filled. The parts must match exactly as specified in the file header (wild characters are not expanded).

Remarks:

ZPRTCI will always print a message to MUNIT, regardless if parts match or not. If LALL is .FALSE. and the parts do not match, the message "No Data Compression Set" will be printed. The information printed is the same as that displayed with the "DC ?" command in DSSUTL.

An example message that might be printed for a file, with LALL set to .TRUE. is:

```
Pathname Parts:  C=PRECIP@
Compression Method:  3;  Repeat + Delta
```

Precision: -2
Compression Software selects allocation space for the Delta scheme.

Pathname Parts: A=NORTH RIVER, C=STAGE
Compression Method: 2; Delta
Precision: -3
Two bytes allocated for each value.

Example:

```
C      Print a file's data compression header.
       CHARACTER CPARTS(6)*32
       LOGICAL LALL
C
C      Attach files, open the DSS file, etc.
       CALL ZOPEN (...
C
       . . .
C
       IF (LALL) THEN
C        Print all of the file's data compression header.
         CALL ZPRTCI (IFLTAB, .TRUE., CPARTS)
       ELSE IF (LPRCIP) THEN
C        Print compression information for precipitation data only.
         DO 20 I=1,6
            CPARTS(I) = ' '
 20      CONTINUE
         CPARTS(3) = 'PRECIP@'
         CALL ZPRTCI (IFLTAB, .FALSE., CPARTS)
       ELSE
         . . .
```

11 Outdated Subroutines

The subroutines documented in this chapter were written for previous versions of DSS and have since been replaced by other routines that are more convenient to use or provide more capability. These routines are still fully supported, but their replacements should be called instead when writing new DSS interface code. In fact, most of these routines rearrange arguments and call their replacements. This chapter is intended as an aid for modifying existing DSS interface code.

11.1 ZFPN - Form DSS Pathname

Purpose:

ZFPN constructs a record pathname from six pathname parts. ZFPN removes leading and trailing blanks from each part, and inserts a slash (/) between each part.

Replaced By:

ZPATH

Calling Sequence:

```
CALL ZFPN (CA, NA, CB, NB, CC, NC, CD, ND, CE, NE, CF, NF, CPATH, NPATH)
```

Declarations:

```
INTEGER NA, NB, NC, ND, NE, NF, NPATH
CHARACTER CA*32, CB*32, CC*32, CD*32, CE*32, CF*32, CPATH*80
```

Argument Description:

CA	Input	A character string containing the A (first) part of the pathname. Up to thirty-two characters may be used in a pathname part. The part may have leading and trailing blanks which will be removed by ZFPN.
NA	Input	The number of characters in CA. This number may include leading and trailing blanks. If the part is null, NA may be set to zero (or CA should contain only blanks).
CB	Input	The B part of the pathname.
NB	Input	The number of characters in CB.
CC	Input	The C part of the pathname.
NC	Input	The number of characters in CC.
CD	Input	The D part of the pathname.
ND	Input	The number of characters in CD.
CE	Input	The E part of the pathname.
NE	Input	The number of characters in CE.

CF	Input	The F part of the pathname.
NF	Input	The number of characters in CF.
CPATH	Output	The completed pathname.
NPATH	Output	The number of characters in CPATH, including slashes.

Remarks:

Each pathname part may contain up to thirty-two characters, and the pathname may contain up to eighty characters (including slashes). If the sum of the parts and slashes is greater than eighty characters, the pathname will be truncated to eighty characters.

A frequent problem encountered is the entrance of control characters (or null characters) in the pathname. This usually occurs when the pathname part is not blanked prior to usage or the part length specified is longer than the dimension of the part. It is good practice to initialize pathname parts with blanks.

Example:

```
C     Form a pathname, reading the A part from the keyboard,
C     and using the location for the B part.
      CHARACTER CA*32, CB*32, CYEAR*4, CPATH*80
C
      READ (5,10,END=100,ERR=900) CA
 10   FORMAT (A)
      CB = CLOC
C
      CALL ZFPN (CA, 32, CB, 32, 'STAGE-DAMAGE', 12, ' ', 0,
     * CYEAR, 4, 'COMPUTED', 8, CPATH, NPATH)
C
      WRITE (6, 20) CPATH(1:NPATH)
 20   FORMAT (' Pathname: ',A)
```

11.2 ZGTDTS - Get Regular-Interval Time Series Data

Purpose:

ZGTDTS retrieves regular-interval time series data from a DSS file. The data retrieved is based on a time window and may cross record boundaries (that is, ZGTDTS may read several records with different D (date) parts to retrieve the data specified).

Replaced By:

ZRRTS and ZRRTSX

Calling Sequence:

```
      CALL ZGTDTS (IFLTAB, CA, NA, CB, NB, CC, NC, CF, NF,
   *  JULS, ISTIME, JULE, IETIME, INTL, DUM, 0,
   *  IOFSET, VALUES, NVALS, CUNITS, CTYPE, ISTAT)
```

Declarations:

```
      INTEGER IFLTAB(600), NA, NB, NC, NF, JULS, ISTIME, JULE
      INTEGER INTL, IETIME, IOFSET, NVALS, ISTAT
      REAL VALUES(NVALS), DUM
      CHARACTER CA*32, CB*32, CC*32, CF*32, CUNITS*8, CTYPE*8
```

On MS-DOS microcomputers, the Julian dates, the time interval and interval offset must be INTEGER*4: INTEGER*4 JULS, JULE, IOFSET, INTL

Argument Description:

IFLTAB	Input/ Output	The DSS work space used to manage the DSS file. This is the same array used in the ZOPEN call.
CA	Input	A character string containing the A part of the pathname. Up to thirty-two characters may be used in a pathname part. The part may have leading and trailing blanks (which will be removed).
NA	Input	The number of characters in CA. This number may include leading and trailing blanks, if desired. If the part is null, NA may be set to zero (or CA should contain only blanks).
CB	Input	The B part of the pathname.
NB	Input	The number of characters in CB.

CC	Input	The C part of the pathname.
NC	Input	The number of characters in CC.
CF	Input	The F part of the pathname.
NF	Input	The number of characters in CF.
JULS	Input	The Julian date of the start of the time window. This is in days since 31DEC1899 (not since the beginning of the current year).
ISTIME	Input	The starting time of the time window, in minutes past midnight (for midnight ISTIME would be 1440, not 0). JULS and ISTIME define the time of the first data value.
JULE	Input	The Julian date of the end of the time window, in days since 31DEC1899.
IETIME	Input	The ending time of the time window in minutes past midnight.
INTL	Input	The time interval of the data in minutes. For hourly data this would be sixty. For monthly data set INTL to 43200.
DUM	Unused	A dummy variable. This used to be the internal header array, which now is returned as parameters (e.g., CUNITS, CTYPE).
0	Unused	Zero. This used to be the length of the internal header array.
IOFSET	Output	The time offset of the data in minutes. (If hourly data is recorded at fifteen minutes past the hour, the offset would be fifteen minutes.) If there is no offset, IOFSET will be returned as zero. (For Version 4, this was buffer space to hold the record. Data is now read directly from the disk into the values array.)
VALUES	Output	The actual data values retrieved. The values in this array are undefined if ISTAT is greater than ten.
NVALS	Input/ Output	As input, this variable must contain the dimension size of VALUES. NVALS is returned with the number of data values read. The array VALUES will contain NVALS real elements.
CUNITS	Output	The units of the data (e.g., FEET).
CTYPE	Output	The type of the data (e.g., PER-AVER).

ISTAT Output A status parameter indicating the success of the operation. If ISTAT is returned with zero, then all the data was successfully read. If ISTAT is returned with a value between one and three, then data was retrieved, but some missing values were detected. If ISTAT is greater than ten, a fatal error occurred, and no data was returned. The possible values are:

ISTAT	**Description**
0	All data retrieved.
1	Some missing data was detected (missing values set to -901.0).
2	Missing record(s) (missing values set to -902.0 for the record not found), but some data was found.
3	Missing record(s) and missing data in the data set, however some data was found.
4	No data found for this time window, but a record was found.
5	No records found (no data is returned).
GT 10	A "fatal" error occurred:
	11 The number of values requested was less than one.
	12 A non-standard time interval was provided.
	15 The starting date or time was not recognized.
	20 The data was not recognized as regular-interval time series.
	24 The pathname given does not meet the regular-interval time series conventions.
	53 The data could not be un-compressed.

Remarks:

ZGTDTS calls ZRRTSX.

CUNITS and CTYPE will contain the units and type for the last record read (when reading several records). If no data was found (ISTAT=4), or a fatal error occurred, CUNITS and CTYPE will be unchanged.

A debug trace will be printed when the message level (MLEVEL) is set to 8 via subroutine ZSET. This trace will print the pathname, dates, times, and other information used by the subroutine.

Example:

```
C       Retrieve the past 50 days of daily data and print them.
C
        INTEGER IFLTAB(600)
```

```
      INTEGER*4 JULS, JULE, INTL, IOFSET
      CHARACTER CA*32, CB*32, CC*32, CF*32
      CHARACTER CDATE*20, CTIME*4, CUNITS*8, CTYPE*8
      REAL VALUES(100)
C
C     Open the DSS file, get the pathname parts, time interval, etc.
      CALL ZOPEN ( ...
C
C
C     Get the current date and time.
      CALL CURTIM ( JULE, IETIME)
      INTL = 1440
C     Decrement it by 49 days (50 values).
      IDUM = INCTIM ( INTL, 0, -49, JULE, IETIME, JULS, ISTIME)
C
C     Now retrieve the data.
      NVALS = 100
      CALL ZGTDTS (IFLTAB, CA, NA, CB, NB, CC, NC, CF, NF,
     * JULS, ISTIME, JULE, IETIME, INTL, DUM, 0, IOFSET,
     * VALUES, NVALS, CUNITS, CTYPE, ISTAT)
      IF (ISTAT.GE.10) GO TO 900
      IF (ISTAT.GE.3)  GO TO 100
C
C     Adjust the starting time to account for any offset.
      CALL ZOFSET ( JULS, ISTIME, INTL, 2, IOFSET)
C
C     Print the data values along with their date and time.
      DO 80 I=1,50
        IDUM = INCTIM (INTL, 0, I-1, JULS, ISTIME, JULE, IETIME)
        CALL JULDAT (JULE, 0, CDATE, NDATE)
        IDUM = M2IHM (IETIME, CTIME)
        WRITE (6,40) CDATE(1:NDATE), CTIME, VALUES(I)
 40     FORMAT (1X,A,2X,A,F10.3)
 80   CONTINUE
```

11.3 ZPTDTS - Put Regular-Interval Time Series Data in a DSS File

Purpose:

ZPTDTS stores regular-interval time series data in a DSS file. The data is based on a time window, and may cross record boundaries (that is, ZPTDTS may write several records with different D (date) parts).

Replaced By:

ZSRTS and ZSRTSX

Calling Sequence:

```
     CALL ZPTDTS (IFLTAB, CA, NA, CB, NB, CC, NC, CF, NF,
   * JULS, ISTIME, JULE, IETIME, INTL, DUM1, IDUM,
   * DUM2, VALUES, NVALS, CUNITS, CTYPE, ISTAT)
```

Declarations:

```
     INTEGER IFLTAB(600), NA, NB, NC, NF, JULS, ISTIME, JULE
     INTEGER INTL, IETIME, IDUM, NVALS, ISTAT
     REAL VALUES(NVALS), DUM1, DUM2
     CHARACTER CA*32, CB*32, CC*32, CF*32, CUNITS*8, CTYPE*8
```

On MS-DOS microcomputers, the Julian dates and time interval must be INTEGER*4:
INTEGER*4 JULS, JULE, INTL

Argument Description:

IFLTAB	Input/ Output	The DSS work space used to manage the DSS file. This is the same array used in the ZOPEN call.
CA	Input	A character string containing the A part of the pathname. Up to thirty-two characters may be used in a pathname part. The part may have leading and trailing blanks (which will be removed).
NA	Input	The number of characters in CA (the dimension). This number may include leading and trailing blanks, if desired. If the part is null, NA may be set to zero (or CA should contain only blanks).
CB	Input	The B part of the pathname.

NB	Input	The number of characters in CB.
CC	Input	The C part of the pathname.
NC	Input	The number of characters in CC.
CF	Input	The F part of the pathname.
NF	Input	The number of characters in CF.
JULS	Input	The Julian date of the start of the time window. This is in days since 31DEC1899 (not since the beginning of the current year).
ISTIME	Input	The starting time of the time window, in minutes past midnight (for midnight ISTIME would be 1440, not zero). JULS and ISTIME define the time of the first data value. Any time offset is implied by these values (e.g., for daily data recorded at 8:00 a.m., set ISTIME to 480, an implied offset of 480 minutes).
JULE	Input	The Julian date of the end of the time window, in days since 31DEC1899.
IETIME	Input	The ending time of the time window in minutes past midnight.
INTL	Input	The time interval of the data in minutes. For hourly data this would be sixty. For monthly data set INTL to 43200.
DUM1	Unused	A dummy variable. In Version 4 this contained the internal header array.
IDUM	Unused	A dummy variable. In Version 4 this was the length of the internal header array.
DUM2	Unused	A dummy variable. In version 4 this was a buffer array to hold the record written to disk. The subroutine now has its own buffers to accomplish this.
VALUES	Input	The actual data values to store. Values in the array that are missing should be set to -901.
NVALS	Input/ Output	As input, this variable must contain the dimension size of VALUES. NVALS is returned with the number of data values stored. The array VALUES must contain NVALS real elements.
CUNITS	Input	The units of the data (e.g., FEET).

CTYPE	Input	The type of the data (e.g., PER-AVER).
ISTAT	Input/ Output	As input, ISTAT is an argument to indicate whether to write over existing data or not. If ISTAT is set to zero, the data provided will always replace any existing data (with the same pathname at the same times). As output, ISTAT is a status parameter indicating the success of the operation. If ISTAT is returned with zero, then all the data was successfully stored. If ISTAT is greater than ten, a fatal error occurred. The possible values are:

As Input:

ISTAT	**Description**
0	Always write over existing data.
-1	Only replace missing data flags in the record (-901).
-2	If all the data are missing data flags (-901), write the record regardless. (Normally, if all the input data values are missing, the record will not be written).
-4	If an input data value is missing (-901), do not allow it to replace an existing data value.

As Output:

ISTAT	**Description**
0	The data was successfully stored.
4	All of the input data provided were missing data flags (-901). No data was stored.
GT 10	A "fatal" error occurred: 11 NVALS is less than one. 12 An illegal time interval was given. 15 The starting date is illegal. 24 The pathname does not meet the regular-interval time series conventions.

Remarks:

ZPTDTS calls ZSRTSX.

The argument NVALS must be set to the dimension of VALUES prior to calling ZPTDTS. NVALS is not necessarily the number of data values to store; the number of data values to store is determined by the time window.

A debug trace may be turned on by setting the message level (MLEVEL) to seven, eight, or nine via subroutine ZSET. Level seven gives information regarding the arguments being passed. The higher levels provide information about the steps taking place inside the subroutine.

If ISTAT is not returned with zero, an error (or warning) message will be written to the standard output, provided the message level is set accordingly. The error messages are explicit. If a fatal error occurs, what the error is, and any relevant information will be printed.

Example:

```
C      Store NVALS data values.
C
       INTEGER IFLTAB(600)
       INTEGER*4 JULS, JULE, INTL, IYMDJL
       CHARACTER CA*32, CB*32, CC*32, CF*32, CTIME*4
       REAL VALUES(1000)
C
C      Open the DSS file, get the pathname parts, etc.
       CALL ZOPEN ( ...
       CALL ZGPNP ( ...
C
C      If the date is in the integer form 12/24/82,
C      convert it to julian.
       JULS = IYMDJL (IYR, IMON, IDAY)
C      Convert the time from 24 hour clock time to minutes.
       ISTIME = IHM2M (CTIME)
C
C      Increment the time by NVALS-1 periods.
       IDUM = INCTIM (INTL, 0, NVALS-1, JULS, ISTIME, JULE, IETIME)
C
C      Now store the data.
       ISTAT = 0
C      As input, NVALS is the dimension of VALUES,
C      not necessarily the number of values.
       NVALS = 1000
       CALL ZPTDTS (IFLTAB, CA, NA, CB, NB, CC, NC, CF, NF,
     * JULS, ISTIME, JULE, IETIME, INTL, DUM, IDUM, DUM,
     * VALUES, NVALS, 'CFS', 'PER-AVER', ISTAT)
       IF (ISTAT.GT.0) GO TO 900
C
```

11.4 ZGIRTS - Get Irregular-Interval Time Series Data

Purpose:

ZGIRTS retrieves irregular-interval time series data from a DSS file. The data retrieved may be based on a time window and can cross record boundaries (i.e., several records with different D parts may be read with one call to ZGIRTS). If no time window is specified, all the data from that record will be retrieved (using the D part for the date).

Replaced By:

ZRITS and ZRITSX

Calling Sequence:

```
     CALL ZGIRTS (IFLTAB, CPATH, NPATH, JULS, ISTIME, JULE, IETIME,
   * DUM1, IDUM1, DUM2, IDUM2, KVALS, DATES, VALUES, NVALS,
   * BDATE, CUNITS, CTYPE, ISTAT)
```

Declarations:

```
     INTEGER IFLTAB(600), NPATH, JULS, ISTIME, JULE, IETIME
     INTEGER IDUM1, IDUM2, KVALS, NVALS, ISTAT
     REAL DATES(KVALS), VALUES(KVALS), BDATE, DUM1, DUM2
     CHARACTER CPATH*80, CUNITS*8, CTYPE*8
```

On MS-DOS microcomputers, the Julian dates must be INTEGER*4:
INTEGER*4 JULS, JULE

Argument Description:

IFLTAB	Input/ Output	The DSS work space used to manage the DSS file. This is the same array used in the ZOPEN call.
CPATH	Input	The pathname of the data to read. The pathname must meet the irregular time-series conventions specified in the HECDSS User's Guide, including a correct E part. With a time window specified, the "D part" (date part) will be ignored; as ZGIRTS will form it internally (there may be several D parts, depending on the time window). If no time window is specified, a correct D part must be provided.
NPATH	Input	The number of characters in CPATH.
JULS	Input	The Julian date of the start of the time window. This is days since 31DEC1899, not since the beginning of the current year.

		If no time window is specified, this argument is ignored (see ISTIME).
ISTIME	Input	The starting time of the time window, in minutes past midnight (for midnight ISTIME would be 1440, not zero). To have no time window set (and read the entire record), set ISTIME to negative two. This will use the D part of the pathname to define the time window.
JULE	Input	The Julian date of the end of the time window in days since 31DEC1899. If no time window is set, this argument is ignored.
IETIME	Input	The ending time of the time window in minutes past midnight. If no time window is set, this argument is ignored.
DUM1	Unused	A dummy variable. In Version 4 this was a buffer array to hold the record to read. The subroutine now has internal buffers to read the record with.
IDUM1	Unused	A dummy variable. In Version 4 this was the dimension of the buffer.
DUM2	Unused	A dummy variable. In Version 4 this was the internal header array, which now is returned as parameters (e.g., CUNITS, CTYPE).
IDUM2	Unused	A dummy variable. In Version 4 this was the length of the internal header array.
KVALS	Input	The dimension of arrays DATES and VALUES, or (if desired) the maximum number of data values to retrieve. No more than KVALS data values will be retrieved.
DATES	Output	An array containing the dates of the data (VALUES), in a one-to-one correspondence. The dates are given in the number of days and fraction of a day from BDATE. On computers where the precision is sufficiently large, BDATE can be added to each element of DATES to produce Julian days and fractions of a day since 31DEC1899. DATES will be returned with NVALS elements.
VALUES	Output	The actual data values retrieved.
NVALS	Output	The number of data values read. Arrays DATES and VALUES will contain NVALS elements.

BDATE	Output	The Julian base date (in days since 31DEC1899), usually equivalent to JULS (with no fractional part). All the dates in array DATES are relative to this date.
CUNITS	Output	The units of the data (e.g., FEET).
CTYPE	Output	The type of the data (e.g., PER-AVER).
ISTAT	Output/	A status parameter indicating the success of the operation. If ISTAT is returned with zero, then the data was successfully read. If ISTAT is greater than ten, a fatal error occurred. The possible values are:

ISTAT	**Description**
0	Successful data retrieval.
1	The number of data values requested (according to the time window) exceeds KVALS. The ITIMES and VALUES arrays will contain KVALS values.
4	No data found (pathname not found). The output arguments are undefined.
GT 10	A "fatal" error occurred: 20 The data was not recognized as irregular-interval time series 21 An internal buffer array is not large enough to read the record. (This will seldom occur as the same array is used to store the data, and the error would be detected at that time.) 24 The pathname does not meet the irregular-interval time series-conventions.

Remarks:

ZGIRTS calls ZRITSX.

The BDATE argument is provided to take care of precision problems on many computers. The complete Julian date and fraction of a day requires precision of nine significant digits (for example a number of 35020.0001). This can be represented by a BDATE of 35000.0 and a DATES(N) value of 20.0001. On machines with a precision of nine or more significant digits, BDATE can be added to each of the DATES elements to compute a total Julian date and fraction. This cannot be done on thirty-two bit machines (e.g., a PC), but can be accomplished on forty-eight bit and larger machines (e.g., HARRIS and CDC). An example of changing this style of date to a standard style is provided in the example following.

CUNITS and CTYPE will contain the units and type for the last record read (when reading several records). If no data was found (ISTAT=4), or a fatal error occurred, CUNITS and CTYPE will be unchanged.

A debug trace may be turned on by setting the message level (MLEVEL) to seven, eight, or nine via subroutine ZSET. Level seven gives information regarding the arguments being passed. The higher levels provide information about the steps taking place inside the subroutine.

Example:

```
C      Retrieve data for the past 60 days and print them.
C
       INTEGER IFLTAB(600)
       INTEGER*4 INTL, JULS, JULE, JUL, JULR, JULB
       REAL DATES(1000), VALUES(1000), BDATE
       CHARACTER CPATH*80, CUNITS*8, CTYPE*8, CDATE*20, CTIME*4
C
C      Open the DSS file and get the pathname parts.
       CALL ZOPEN ( ...
C
C      Get the current julian date and time.
       CALL CURTIM (JULE, IETIME)
       INTL = 1440
C      Decrement it by 60 days (from midnight).
       IDUM = INCTIM ( INTL, 0, -60, JULE, IETIME, JULS, ISTIME)
       ISTIME = 1440
C
C      Retrieve the data.
       CALL ZGIRTS (IFLTAB, CPATH, NPATH, JULS, ISTIME, JULE, IETIME,
     * DUM1, IDUM1, DUM2, IDUM2, 1000, DATES, VALUES, NVALS, BDATE,
     * CUNITS, CTYPE, ISTAT)
C
C      Check for errors.
       IF (ISTAT.EQ.4) GO TO 100
       IF (ISTAT.GE.10) GO TO 900
C
C      Print the data values.
       WRITE (6,20) CUNITS, CTYPE
 20    FORMAT (' Units: ',A,',  Type: ',A)
C
       JULB = INT4(BDATE)
       DO 60 I=1,NVALS
C        Convert julian dates to standard.
         JULR = INT4(DATES(I))
         JUL = JULB + JULR
         FRACT = DATES(I) - REAL(JULR)
         RMIN = (FRACT * 1440.) + 0.6
         MIN = INT(RMIN)
C        Convert to standard date and time.
         CALL JULDAT (JUL, 0, CDATE, NDATE)
```

```
          IDUM = M2IHM (MIN, CTIME)
    C     Print the value.
          WRITE (6,40) CDATE(1:NDATE), CTIME, VALUES(I)
     40   FORMAT (1X,A,', ',A,': ',F8.2)
     60 CONTINUE
```

11.5 ZPIRTS - Put Irregular-Interval Time Series Data

Purpose:

ZPIRTS stores irregular-interval time series data in a DSS file. The data is stored based upon an implied time window which can cross record boundaries. The time window is implied by the date and times of the first and last values in the time array.

Irregular-interval time series data is stored with times to the nearest minute. Data for times of less than a minute cannot be stored with this convention. The times of the data must be in ascending order, and no value may have the same exact time as another (you cannot have two data points for the same time in a record).

Replaced By:

ZSITS and ZSITSX

Calling Sequence:

```
      CALL ZPIRTS (IFLTAB, CPATH, NPATH, DUM1, IDUM1, DUM2,
     *   IDUM2, DATES, VALUES, NVALS, BDATE, CUNITS, CTYPE,
     *   INFLAG, ISTAT)
```

Declarations:

```
INTEGER IFLTAB(600), NPATH, NVALS, INFLAG, ISTAT, IDUM1, IDUM2
REAL DATES(NVALS), VALUES(NVALS), BDATE, DUM1, DUM2
CHARACTER CPATH*80, CUNITS*8, CTYPE*8
```

Argument Description:

IFLTAB	Input/ Output	The DSS work space used to manage the DSS file. This is the same array used in the ZOPEN call.
CPATH	Input	The pathname of the data to store. The pathname must meet the irregular-interval time series conventions specified in the HECDSS Users Guide, including a correct E part. The D part (date part) is ignored, as ZPIRTS will form it internally.
NPATH	Input	The number of characters in CPATH.
DUM1	Unused	A dummy variable. In Version 4 this was a buffer array to hold the record to store. The subroutine now has internal buffers to accomplish this.
IDUM1	Unused	A dummy variable. In Version 4 this was the dimension of the buffer.

DUM2	Unused	A dummy variable. In Version 4 this was the internal header array.
IDUM2	Unused	A dummy variable. In Version 4 this was the length of the internal header array.
DATES	Input	The array containing the dates of the data (VALUES), in a one-to-one correspondence. The dates must be given in days and fraction of a day from BDATE, where the addition of BDATE and DATES(I) is the date and time of the respective data value. On computers where the precision is sufficiently large, BDATE can be set to zero, and each element of DATES may be the total Julian day and fraction of the day since 31DEC1899. DATES should contain NVALS elements.
VALUES	Input	The data values to store.
NVALS	Input	The number of data values to store. Arrays DATES and VALUES must contain NVALS valid elements.
BDATE	Input	The base date, which when added with each element of DATES will give the total Julian day and fraction of a day for the respective data. All values in array DATES should be relative to this value. (See remarks).
CUNITS	Input	The units of the data (e.g., FEET).
CTYPE	Input	The type of the data (e.g., PER-AVER).

INFLAG	Input	INFLAG is a flag to indicate whether the data should be replaced or merged with existing data. Replace will replace all the data between the implied time window (time of first and last data). Merge will combine the data with the data already stored. (Merging data replaces data occurring at the same time and inserts data at new times.)
		INFLAG = 0 to merge data
		INFLAG = 1 to replace data

ISTAT	Output	A status parameter indicating the success of the operation. If ISTAT is returned with zero, then all the data was successfully stored. If ISTAT is greater than ten, a fatal error occurred. The possible values are:

ISTAT	**Description**
0	The data was successfully stored.
4	No data was given to store (NVALS was zero).
GT 10	A "fatal" error occurred:
	21 An internal buffer array is not large enough to store this number of data values. If this

	error occurs, the time-block identified by in the "E part" of the pathname spans too long of a time period, and holds more data values than the internal buffers can accommodate. The time-block should be changed to the next lower size (e.g., from "IR-MONTH" to "IR-DAY").
24	The pathname does not meet the irregular-interval time series conventions.
30	The times associated with the data values are not in an ascending order, or two values occur at the same time.

Remarks:

ZPIRTS calls ZSITSX.

The BDATE argument is provided to take care of precision problems on many computers. The complete Julian date and fraction of a day require a precision of nine significant digits (for example a number of 35020.0001). This can be represented by a BDATE of 35000.0 and a DATES(N) value of 20.0001. On machines with a precision of nine or more significant digits, BDATE can be set to zero, and each of the DATES elements may be a total Julian date and fraction. Typically this cannot be done on thirty-two bit machines (e.g., a PC), but can be accomplished on forty-eight bit and larger machines (e.g., HARRIS and CDC). An example of obtaining this style of date from a standard style is provided in the example following.

With reference to INFLAG and data already present in the DSS file, replace will replace all the data within the implied time window, while merge will combine the two data sets, only replacing those values that occur at exactly the same time (within one minute of significance). INFLAG has no meaning for a new record. Usually the replace mode is used for editing data, and the merge mode is used for adding new data to the record.

A debug trace may be turned on by setting the message level (MLEVEL) to nine via subroutine ZSET.

Example:

```
C     Read data from an ASCII file and store in the DSS file.
      INTEGER IFLTAB(600), IBF(5), IEF(5), ILF(5)
      INTEGER*4 JUL
      CHARACTER CPATH*80, CLINE*80, CUNITS*8, CTYPE*8
      REAL DATES(1000), VALUES(1000)
C
C     Open the DSS file and get the pathname parts.
      CALL ZOPEN ( ...
C
```

```
      READ (5,20) CUNITS, CTYPE
 20   FORMAT (A,A)
      NVALS = 0
C
 100  CONTINUE
      IF (NVALS.GE.1000) GO TO 200
      READ (5,20,END=200) CLINE
C
C     Parse the line.
      CALL PARSLI (CLINE, 5, NFIELD, IBF, IEF, ILF)
      IF (NFIELD.NE.3) GO TO 900
C
C     The date should be in the first field, the time in
C     the second, and the data in the third field.
      NVALS = NVALS + 1
      CALL DATJUL (CLINE(IBF(1):IEF(1)), JUL, IERR)
      IF (IERR.NE.0) GO TO 900
C     Set BDATE to the date of the first value.
      IF (NVALS.EQ.1) BDATE = REAL(JUL)
C
      NTIME = IHM2M (CLINE(IBF(2):IEF(2)))
      IF (NTIME.LT.0) GO TO 900
C
      DATES(NVALS) = (REAL(JUL) - BDATE) + (REAL(NTIME)/1440.)
      VALUES (NVALS) = XREAL (CLINE, IBF(3), ILF(3), IERR)
      IF (IERR.NE.0) GO TO 900
C
      GO TO 100
C
 200  CONTINUE
C     Now store the data.
      IF (NVALS.LE.0) GO TO 800
      CALL ZPIRTS ( IFLTAB, CPATH, NPATH, DUM, IDUM, DUM, IDUM,
     * DATES, VALUES, NVALS, BDATE, CUNITS, CTYPE, 0, ISTAT)
      IF (ISTAT.NE.0) GO TO 900
```

11.6 ZGTPFD - Get Paired Function Data

Purpose:

ZGTPFD retrieves paired function data from a DSS file.

Replaced By:

ZRPD

Calling Sequence:

```
      CALL ZGTPFD (IFLTAB, CPATH, NPATH, NORD, NCURVE, IHORIZ
   *  C1UNIT, C2UNIT, C1TYPE, C2TYPE, CLABEL, KLABEL, NLABEL,
   *  DUM, IDUM1, IDUM2, VALUES, KVALS, NVALS, ISTAT)
```

Declarations:

```
      INTEGER IFLTAB(600), NPATH, NORD, NCURVE, IHORIZ
      INTEGER KLABEL, NLABEL, IDUM1, IDUM2, KVALS, NVALS, ISTAT
      REAL VALUES(KVALS)
      CHARACTER CPATH*80, C1UNIT*8, C2UNIT*8, C1TYPE*8, C2TYPE*8
      CHARACTER CLABEL(KLABEL)*12
```

Argument Description:

IFLTAB	Input/ Output	The DSS work space used to manage the DSS file. This is the same array used in the ZOPEN call.
CPATH	Input	The pathname of the data to retrieve. The pathname must meet the paired function data conventions.
NPATH	Input	The number of characters in CPATH.
NORD	Output	The number of ordinates read (number of points per curve). Each curve stored in a single record must have the same number of ordinates.
NCURVE	Output	The number of curves retrieved.
IHORIZ	Output	The variable number to appear on the horizontal axis for plotting (one for first variable, two for second).
C1UNIT	Output	The units of the first variable (e.g., 'FEET', or 'PERCENT').
C2UNIT	Output	The units of the second variable.

C1TYPE	Output	The type of data for the first variable. The following types are recognized by the DSS utility programs: UNT Untransformed LOG Logarithmic - data expressed as logarithms PROB Probability - data expressed in percent
C2TYPE	Output	The type of data for the second variable.
CLABEL	Output	The labels for each curve. For example, if an ELEVATION-DAMAGE function is retrieved containing residential, agricultural and commercial damage, then CLABEL might be returned as: CLABEL(1) = 'RESIDENTIAL ' CLABEL(2) = 'AGRICULTURAL' CLABEL(3) = 'COMMERCIAL ' For this example, NCURVE would be returned with three, and CLABEL should be dimensioned to at least three.
KLABEL	Input	The dimension of CLABEL. No more than KLABEL labels will be placed into CLABEL.
NLABEL	Output	The number of labels read.
DUM	Unused	A dummy variable. In Version 4 this contained the internal header array.
IDUM1	Unused	A dummy variable. In Version 4 this was the dimension of the internal header array.
IDUM2	Unused	A dummy variable. In Version 4 this was the length of the internal header array.
VALUES	Output	The data values retrieved. The first NORD elements in VALUES correspond to the first variable. The data for the second variable begins at element NORD+1.
KVALS	Input	The dimension of array VALUES. VALUES must be dimensioned to at least: KVALS = (NCURVE + 1) * NORD
NVALS	Output	The number of values retrieved.
ISTAT	Output	A status parameter indicating the success of the operation. If ISTAT is returned with zero, then the data was successfully read. The possible values are:

ISTAT	Description
0	Successful data retrieval.
-1	The record does not exist. The output arguments are undefined.
1	The dimension of VALUES (KVALS) was not large enough to retrieve all the data. Only KVALS values returned; the curves are incomplete.
20	The record is not paired data.

Remarks:

ZGTPFD calls ZRPD.

Up to fifty curves (with the same ordinates) can be stored in one record. The maximum number of labels is also fifty. Either all curves will have a label, or no curves will have labels. If the VALUES array is dimensioned smaller than the number of data values in the record, only the first KVALS values will be retrieved.

A debug trace will be printed when the message level (MLEVEL) is set to seven (or above) via subroutine ZSET.

Points can be located within a singly dimension array by the following example:

```
C     To print the data (as X, Y1, Y2, Y3, ...):
      DO 20 I=1,NORD
         WRITE (6,10) (VALUES(J),J=I,NVALS,NORD)
 10      FORMAT (' X:',F8.2,',  Y(s):',50(2X,F8.2))
 20   CONTINUE

C     To transform the data into a doubly dimensioned array:
      IPOS = 0
      DO 20 I=1,NCURVE+1
         DO 20 J=1,NORD
            IPOS = IPOS + 1
            CURVE(J,I) = VALUES(IPOS)
 20   CONTINUE
```

Example:

```
C     Retrieve paired data from a DSS file.
      PARAMETER (KVALS=900, KLABEL=8)
      INTEGER IFLTAB(600)
      REAL VALUES(KVALS)
      CHARACTER CPATH*80, C1UNIT*8, C2UNIT*8, C1TYPE*8, C2TYPE*8,
     * CLABEL(KLABEL)*12
C
```

```
C       Open the DSS file.
        CALL ZOPEN( . . .
C
C       Get the pathname.
        CALL ZPATH ( . . .
C
C
C       Retrieve the data.
        CALL ZGTPFD (IFLTAB, CPATH, NPATH, NORD, NCURVE, IHORIZ
     *  C1UNIT, C2UNIT, C1TYPE, C2TYPE, CLABEL, KLABEL, NLABEL,
     *  DUM, IDUM1, IDUM2, VALUES, KVALS, NVALS, ISTAT)
        IF (ISTAT.NE.0) GO TO 900
C
C       Write the record's pathname.
        WRITE (6,20) CPATH(1:NPATH)
 20     FORMAT (' Record Pathname: ',A)
C
C       Write the label information (if there are labels).
        IF (NLABEL.GE.1) THEN
          WRITE (6,30)
 30       FORMAT (' Curve Labels:')
          DO 50 I=1,NLABEL
             WRITE (6,40) I, CLABEL(I)
 40          FORMAT (' Curve',I3,' Label: ',A)
 50       CONTINUE
        ENDIF
C
C       Write the data (as point, X, Y1, Y2, Y3, ...).
        DO 80 I=1,NORD
          WRITE (6,60) I, (VALUES(J),J=I,NVALS,NORD)
 60       FORMAT (' Point',I4,';  X:',F8.2,',  Y(s):',50(2X,F8.2))
 80     CONTINUE
C
```

Example results for an ELEVATION-DAMAGE function having two damage categories and eighteen ordinates:

Input:
CPATH = /JAMES RIVER/DR1/ELEVATION-DAMAGE//1980/PLAN B/
NPATH = 48
KLABEL = 10
KVALS = 1000

Output:
NORD = 18
NCURVE = 2
IHORIZ = 2

```
C1UNIT = 'FEET'
C1TYPE = 'UNT'
C2UNIT = '$1000'
C2TYPE = 'UNT'
NVALS = 54
LABEL = .TRUE.
CLABEL(1) = 'S.F. RES'
CLABEL(2) = 'COMMERCIAL'
ISTAT  = 0
```

The VALUES array contains all of the data:
VALUES(1) through VALUES(18) contain the ELEVATION data.
VALUES(19) through VALUES(36) contain DAMAGE data for "S.F. RES".
VALUES(37) through VALUES(54) contain DAMAGE data for "COMMERCIAL".

11.7 ZPTPFD - Put Paired Function Data

Purpose:

ZPTPFD stores paired function data in a DSS file.

Replaced By:

ZSPD

Calling Sequence:

```
   CALL ZPTPFD (IFLTAB, CPATH, NPATH, NORD, NCURVE, IHORIZ,
*  C1UNIT, C2UNIT, C1TYPE, C2TYPE, CLABEL, KLABEL, NLABEL,
*  DUM, IDUM1, IDUM2, VALUES, KVALS, NVALS, IPLAN, ISTAT)
```

Declarations:

```
INTEGER IFLTAB(600), NPATH, NORD, NCURVE, IHORIZ
INTEGER KLABEL, NLABEL, KVALS, NVALS, IPLAN, ISTAT
INTEGER IDUM1, IDUM2
REAL VALUES(KVALS), DUM
CHARACTER CPATH*80, C1UNIT*8, C2UNIT*8, C1TYPE*8, C2TYPE*8
CHARACTER CLABEL(KLABEL)*12
```

Argument Description:

IFLTAB	Input/ Output	The DSS work space used to manage the DSS file. This is the same array used in the ZOPEN call.
CPATH	Input	The pathname of the data to store. The pathname must meet the paired function data conventions.
NPATH	Input	The number of characters in CPATH.
NORD	Input	The number of ordinates (number of points per curve). Each curve to store in a single record must have the same number of ordinates.
NCURVE	Input	The number of curves to store in this record.
IHORIZ	Input	The variable number to appear on the horizontal axis for plotting (one for first variable, two for second).
IHORIZ	Input	The variable number to appear on the horizontal axis for plotting (one for first variable, two for second).

C1UNIT	Input	The units of the first variable (e.g., 'FEET', or 'PERCENT').
C2UNIT	Input	The units of the second variable.
C1TYPE	Input	The type of data for the first variable. The following types are recognized by the DSS utility programs: UNT Untransformed LOG Logarithmic - data expressed as logarithms PROB Probability - data expressed in percent
C2TYPE	Input	The type of data for the second variable.
CLABEL	Input	A optional character array with labels corresponding to each curve. For example, if an ELEVATION-DAMAGE function is to be stored containing residential, agricultural and commercial damage, then CLABEL might be as follows: CLABEL(1) = 'RESIDENTIAL ' CLABEL(2) = 'AGRICULTURAL' CLABEL(3) = 'COMMERCIAL '
KLABEL	Input	The dimension of CLABEL.
NLABEL	Input	The number of curve labels to store. If no labels are to be stored, set this to zero. If labels are supplied, NLABEL must be equal to NCURVE.
DUM	Unused	A dummy variable. In Version 4 this contained the internal header array.
IDUM1	Unused	A dummy variable. In Version 4 this was the dimension of the internal header array.
IDUM2	Unused	A dummy variable. In Version 4 this was the length of the internal header array.
VALUES	Input	The data values to store. The first NORD elements in VALUES correspond to the first variable (the X axis). The data for the second variable must begin at element NORD+1 (the Y axis).
KVALS	Input	The dimension of VALUES.
NVALS	Output	The number of values stored.
IPLAN	Input	An argument indicating whether to write over existing data or not:

IPLAN	**Description**
0	Always write the record to the file.
1	Only write the record if it is new (i.e., no record exists with that pathname).
2	Only write the data if the record already existed in the file.

ISTAT Output A status parameter indicating the success of the operation. If ISTAT is returned with zero, then the data was successfully stored, otherwise an error occurred. The possible values are

ISTAT	**Description**
0	The data was successfully stored.
-1	IPLAN requested that the record be written only if it was new, but the file already contained a record with the pathname supplied.
-2	IPLAN requested that the record be written only if it already existed, but the pathname supplied was not found.
-3	The pathname does not meet the paired data conventions).
-4	The number of ordinates is less than one.
-5	NCURVE is less than one or greater than 50.

Remarks:

ZPTPFD calls ZSPD.

Up to fifty curves (with the same ordinates) can be stored in one record. The maximum number of labels is also fifty. Either all curves will have a label, or no curves will have labels.

A debug trace will be printed if the message level (MLEVEL) is set to seven (or above) via subroutine ZSET.

Unless the number of data points for the curve(s) is known prior to obtaining them (for example, if you are reading them from an external file), the data usually must be read into a buffer, then reorganized into a singly dimensioned array before storing with ZSPD. Points can be converted from a doubly dimensioned array into a singly dimensioned array by the following example:

```
C
C     The data has been read into array CURVE as X, Y1, Y2, Y3, ...
      IPOS = 0
      DO 20 I=1,NCURVE+1
      DO 20 J=1,NORD
        IPOS = IPOS + 1
        VALUES(IPOS) = CURVE(J,I)
 20   CONTINUE
```

Example:

```
C       Read (a) Curve(s) from an external file, then store it in DSS.
C       Up to 10 curves (in one record) can be stored by this routine.
C       The external file contains data in the form:
C       X, Y1, Y2, . . .
C       X, Y1, Y2, . . .
C       END
C
        PARAMETER (KVALS=1000, KLABEL=10)
        INTEGER IFLTAB(600), NORD, NCURVE, IHORIZ
        INTEGER ISTAT, IBF(20), IEF(20), ILF(20)
        REAL VALUES(KVALS), CURVES(300,11)
        CHARACTER CPATH*80, C1UNIT*8, C2UNIT*8, C1TYPE*4, C2TYPE*4
        CHARACTER CLABEL(KLABEL)*12, CNAME*64, CLINE*80
C
C       Open the DSS file.
        CALL ZOPEN (IFLTAB, CNAME, ISTAT)
        IF (ISTAT.NE.0) GO TO 900
C
C       Get the pathname.
        CALL ZPATH (. . .
C
C       Get the number of Curves, IHORIZ.
        READ (5,*) NCURVE, IHORIZ
C
C       Get the data units and type.
        READ (5,20) C1UNIT, C1TYPE, C2UNIT, C2TYPE
C
C       Read the label information.
        DO 40 I=1,NCURVE
          READ (5,30) CLABEL(I)
 40     CONTINUE
C
C       Read the data (as X, Y1, Y2, Y3, ...).
        NORD = 0
 50     CONTINUE
        READ (5,60,END=200) CLINE
 60     FORMAT (A)
C
C       Did we reach the end of the data yet?
        IF (INDEX(CLINE,'END').GT.0) GO TO 100
C
C       Parse the line.
        CALL PARSLI (CLINE, 20, NFIELD, IBF, IEF, ILF)
        IF (NFIELD.NE.NCURVE+1) GO TO 900
C       Place the data in the curves array.
```

```
      NORD = NORD + 1
      DO 80 I=1,NFIELD
         CURVES(NORD,I) = XREAL (CLINE, IBF(I), ILF(I), IERR)
         IF (IERR.NE.0) GO TO 900
 80   CONTINUE
C
C     Go back and read the next value.
      GO TO 50
C
 100  CONTINUE
C     All the data has been read.  Transfer the data into
C     a singly dimensioned array.
      IPOS = 0
      DO 120 I=1,NCURVE+1
      DO 120 J=1,NORD
         IPOS = IPOS + 1
         VALUES(IPOS) = CURVES(J,I)
 120  CONTINUE
C
C     Store the data.
      NLABEL = NCURVE
      CALL ZPTPFD (IFLTAB, CPATH, NPATH, NORD, NCURVE, IHORIZ,
    * C1UNIT, C2UNIT, C1TYPE, C2TYPE, CLABEL, KLABEL, NLABEL,
    * DUM, IDUM1, IDUM2, VALUES, KVALS, NVALS, 0, ISTAT)
      IF (ISTAT.NE.0) GO TO 900
```

Example results for storing an ELEVATION-DAMAGE function having two damage categories ("S.F. RES" and "COMMERCIAL") and five ordinates:

Elevation	S.F. Res Damage	Commercial Damage
500.0	0.0	0.0
502.0	25.8	0.0
504.0	51.2	323.4
506.0	93.8	655.7
508.0	137.9	809.1

Input:

```
CPATH = /JAMES RIVER/DR1/ELEVATION-DAMAGE//1980/PLAN B/
NORD = 18
NCURVE = 2
IHORIZ = 2
C1UNIT = 'FEET'
C1TYPE = 'UNT'
C2UNIT = '$1000'
C2TYPE = 'UNT'
LABEL = .TRUE.
```

```
CLABEL(1) = 'S.F. RES'
CLABEL(2) = 'COMMERCIAL'
NHEADU = 0
IPLAN  = 0
```

The VALUES array contains all of the ELEVATION-DAMAGE data:
VALUES(1) through VALUES(5) contain the ELEVATION data.
VALUES(6) through VALUES(10) contain DAMAGE data for "S.F. RES".
VALUES(11) through VALUES(15) contain DAMAGE data for "COMMERCIAL".

For example:

```
VALUES(1)  = 500.0
VALUES(2)  = 502.0
...
VALUES(5)  = 508.0
VALUES(6)  =   0.0
VALUES(7)  =  25.8
...
VALUES(10) =  137.9
VALUES(11) =    0.0
VALUES(12) =    0.0
...
VALUES(15) =  809.1
```

11.8 ZOPCAT - Open a Catalog File

Purpose:

ZOPCAT opens a DSS file's catalog file. If the catalog file does not exist, ZOPCAT can create it. If the file does exist, ZOPCAT returns the number of records in the catalog. ZOPCAT cannot open the condensed catalog file.

Replaced By:

ZOPNCA

Calling Sequence:

```
      CALL ZOPCAT (CDSSFI, CATFIL, ICUNIT, LOPEN, LCATLG,
     * LCREAT, NRECS)
```

Declarations:

```
      INTEGER ICUNIT, NRECS
      CHARACTER CDSSFI*64, CATFIL*64
      LOGICAL LOPEN, LCATLG, LCREAT
```

On MS-DOS microcomputers, NRECS must be INTEGER*4: INTEGER*4 NRECS

Argument Description:

CDSSFI	Input	The name of the DSS file whose catalog file is to be opened.
CATFIL	Output	The name of the catalog file.
ICUNIT	Input	The unit number to open the catalog file with. (Most DSS utility programs use Unit 12 for the catalog file).
LOPEN	Input	A logical variable indicating the status of the open. LOPEN will be .TRUE. if the catalog file was successfully opened (otherwise it will be .FALSE.)
LCATLG	Output	A logical variable returned as .TRUE. if the file opened is a valid catalog file. If LCATLG is .FALSE., ZCAT should be called to generate a catalog of the DSS file.
LCREAT	Input	A logical flag indicating whether the catalog file should be created, if it does not exist. When set to .TRUE., the file will be created.

NRECS	Output	The number of records in the catalog file. This is the number shown in the catalog header.

11.9 ZCATLG - Catalog a DSS File

Purpose:

ZCATLG generates a catalog (or listing) of the record pathnames in a DSS file. The catalog may be sorted by pathname parts. ZCATLG can create a selective catalog by matching pathname parts. The selective catalog can be created from a current catalog (reducing resources), or directly from the DSS file. The catalog file must be opened externally by subroutine ZOPNCA.

ZCATLG cannot produce a condensed catalog.

Replaced By:

ZCAT

Calling Sequence:

```
      CALL ZCATLG (IFLTAB, ICUNIT, INUNIT, CINSTR, IBEG, NINSTR,
    * LABREV, LSORT, NRECS)
```

Declarations:

```
      INTEGER IFLTAB(600), ICUNIT, INUNIT, IBEG, NINSTR, NRECS
      CHARACTER CINSTR*(*)
      LOGICAL LABREV, LSORT
```

On MS-DOS microcomputers, NRECS must be INTEGER*4: INTEGER*4 NRECS

Argument Description:

IFLTAB	Input/ Output	The DSS work space used to manage the DSS file. This is the same array used in the ZOPEN call.
ICUNIT	Input	The unit number of file where the catalog is to be written. If a new catalog is to be made, this should be the unit number of the catalog file. If a selective catalog is to be produced from an existing catalog, this unit should probably be attached to a scratch file.
INUNIT	Input	The input catalog unit number. If a new catalog is to be made, this must be set to zero. If a selective catalog is to be produced from an existing catalog, this is the unit number of the existing catalog. If INUNIT is non-zero, the DSS file will not be cataloged.
CINSTR	Input	A character string containing any instructions for generating the catalog, such as the sort order or selective pathname parts. For

		example, if CINSTR is 'O=FB, C=FLOW', the catalog will be sorted in the pathname part order of FBACED, and only those pathnames with a C part of "FLOW" will be cataloged. CINSTR is usually a portion of the input line of the program. If no special instructions are given, set this to blank (' ').
IBEG	Input	The beginning position in CINSTR (all characters prior to IBEG are ignored).
NINSTR	Input	The number of characters in CINSTR to use, starting at IBEG. If there are no instructions, set NINSTR to zero.
LABREV	Input	A logical flag indicating whether an abbreviated catalog should be produced. If set to .TRUE., an abbreviated catalog will be generated, otherwise the standard catalog will be produced.
LSORT	Input	A logical flag indicating whether the pathnames should be sorted. When LSORT is set to .TRUE., the pathnames are sorted (this takes longer than an unsorted catalog).
NRECS	Output	NRECS(Output) The number of records cataloged. This number will be the same as the reference number for the last pathname in the catalog file.

Remarks:

ZCATLG calls ZCAT.

A description of the catalog may be found in the "HECDSS User's Guide and Utility Program Manuals", Overview section. Additional information may also be found in the DSSUTL documentation, located in the same publication. Information about the selective catalog is located in Chapter 5 of the DSSUTL documentation.

The fastest catalog that can be generated is an unsorted abbreviated catalog. In this procedure, pathnames are just copied from the internal DSS address tables to the catalog. In a regular extended catalog, each record must be accessed to obtain the program name, date and time, etc.

After the catalog has been generated, it may be displayed on the screen by reading directly from the catalog. ZCATLG should not be used to display the catalog on the screen (do not set ICUNIT to standard output).

Pathnames may be read from the catalog file with subroutine ZRDCAT, which is a general catalog reading routine, or by subroutine ZRDPAT, which obtains pathnames based on their reference number. If you desire to display an abbreviated catalog and a regular catalog already exists, use subroutine ZRDPAT to read the pathnames from the catalog.

Units 66, 67, 68, and 69 are used for sorting. On Harris computers, work files W2, W3, U1, and U2 are used for sorting (and their contents destroyed). On other computers the files dsssort.in, dsssort.out, and dsssort.tmp are used then deleted.

A status line with the percent complete can be displayed on the screen by setting 'CAST' to 'ON' with ZSET before calling ZCATLG. The message unit must be connected to the screen to display a status line.

A "catalog map" may be generated by ZCATLG when creating a new catalog. A catalog map is a listing of the pathnames only (no title or reference numbers), which is useful for creating an input file of pathnames for some programs. This option is initiated by setting the map options in subroutine ZSET. The map file must be opened and its unit number passed to ZSET through the MAPUNT parameter, then the ZSET MAP parameter must be set to 'ON'. The catalog map is only created when a new catalog is generated. Be sure to call ZSET with MAP set to 'OFF' after the map has been made.

11.10 ZRDPN - Read Pathnames from a Catalog File by Reference Number

Purpose:

ZRDPN reads pathnames from a catalog file according to their reference numbers. ZRDPN may be used in a loop to obtain a set of pathnames, or it can read a single pathname.

Replaced By:

ZRDPAT

Calling Sequence:

```
CALL ZRDPN (ICUNIT, IPOS, INUMB, CPATH, NPATH)
```

Declarations:

```
INTEGER ICUNIT, IPOS, INUMB, NPATH
CHARACTER CPATH*80
```

On MS-DOS microcomputers, IPOS and INUMB must be INTEGER*4:

```
INTEGER*4 IPOS, INUMB
```

Argument Description:

ICUNIT	Input	The unit number of the catalog file.
IPOS	Input/ Output	A file position indicator used by ZRDPN. When first reading from the catalog, the file should be rewound and IPOS set to zero. (The calling program must always set IPOS to zero when the catalog is rewound.) Upon reaching the end of the catalog file, IPOS will be set to 100,000, and no pathname will have been read (CPATH would be undefined).
INUMB	Input/ Output	The catalog reference number of the pathname to read. When the end of the catalog file is reached, INUMB will be returned with the reference number of the last pathname in the catalog. If IPOS is less than or equal to INUMB on input, ZRDPN will return the next pathname, and its reference number, in the catalog.
CPATH	Output	The pathname corresponding to the reference number INUMB.
NPATH	Output	The number of characters in the pathname.

Remarks:

ZRDPN calls ZRDPAT.

The catalog file must be rewound and IPOS set to zero before calling ZRDPN to retrieve a (set of) pathname(s). ZRDPN only can search for pathnames in a forward direction; NUMB must always be greater than IPOS. Thus, if a sequence of reference numbers for pathnames to be retrieved is "12, 18, 9, 20", then the catalog has to be rewound and IPOS set to zero after reading pathname eighteen before pathname nine will be found. It is more efficient to sort the reference numbers in ascending order prior to calling ZRDPN.

If INUMB is less than or equal to IPOS on input, ZRDPN will read the next pathname in the catalog and return its reference number as INUMB. Thus, the entire catalog file can be read by rewinding the file, setting IPOS and INUMB both to zero, and then calling ZRDPN until IPOS is 100,000. In this case INUMB does not need to be reset by the program each time ZRDPAT is called.

Example 1:

```
C     Read a single pathname (e.g., INUMB=24).
      CALL ZOPNCA (...
C
      REWIND 12
      IPOS = 0
      INUMB = 24
      CALL ZRDPN (12, IPOS, INUMB, CPATH, NPATH)
C     If IPOS=100,000, then pathname 24 was not found.
      IF (IPOS.GE.100000) GO TO 900
```

Example 2:

```
C     Read a series of pathnames from 10 through 60.
      REWIND 12
      IPOS = 0
      INUMB = 9
10    CONTINUE
      INUMB = INUMB + 1
      CALL ZRDPN (12, IPOS, INUMB, CPATH, NPATH)
      IF (IPOS.GE.100000) GO TO 900
      WRITE (6,20) INUMB, CPATH(1:NPATH)
 20   FORMAT (1X,I6,2X,A)
      IF (INUMB.LT.60) GO TO 10
```

Example 3:

```
C     Read the set of pathnames whose reference numbers are
C     contained in the array NUMBS (e.g., 8, 12, 15, 9, 20, 13).
```

```
C
      REWIND 12
      IPOS = 0
      DO 20 I=1,JNUMBS
        INUMB = NUMBS(I)
C       Are the numbers in ascending order?
C       If not, rewind the catalog and reset IPOS.
        IF (INUMB.LE.IPOS) THEN
           REWIND 12
           IPOS = 0
        ENDIF
        CALL ZRDPN (12, IPOS, INUMB, CPATH, NPATH)
        IF (IPOS.GE.100000) GO TO 900
        WRITE (6,10) INUMB, CPATH(1:NPATH)
10      FORMAT (1X,I6,2X,A)
20    CONTINUE
```

Appendix A Example Application

The following example shows how a program may be interfaced with the DSS software for retrieving and storing data. This example uses regular-interval time series data. Other types of data follow a similar procedure. The steps in this example are summarized in Chapter 1.

```
      CHARACTER CNAME*64, CDSSIN*64, CDSSOT*64, CLINE*80
      INTEGER IFLTAB(600)
      LOGICAL LDSSIN
      DATA LDSSIN /.FALSE./
C     . . .
C
      Open unit 6 to the standard output via subroutine ATTACH.
      CALL ATTACH ( 6, 'OUTPUT', 'STDOUT', ' ', CNAME, ISTAT)
C
C     Obtain the names of the DSS input and DSS output files.
C     ATTACH passes information from the execution line to the program.
C     The following code allows either one file specified for both the
C     input and output DSS file, or a separate DSS file for input and
C     for output.
      CNAME = ' '
      CALL ATTACH (IDUM, 'DSSFILE', ' ', 'NOP', CNAME, ISTAT)
      CALL ATTACH (IDUM, 'DSSIN', CNAME, 'NOP', CDSSIN, ISTAT)
      CALL ATTACH (IDUM, 'DSSOUT', CNAME, 'NOP', CDSSOT, ISTAT)
C
C
C     READ INPUT.
C
C     As the input file is being read in, look for either a "ZR",
C     or a "ZW" card to trigger access to the DSS file.
      READ (5, 20, END=800, ERR=900) CLINE
 20   FORMAT (A)
C
      IF (CLINE(1:2).EQ.'ZR') THEN
         CALL DSSIN (CDSSIN, CLINE, IFLTAB)
C        Remember that this DSS file was opened.
         LDSSIN = .TRUE.
      ELSE IF (CLINE(1:2).EQ.'ZW') THEN
C        For DSS output, just save the ZW line, until the results
C        are computed (subroutine DSSOUT will use CLINE).
         CALL SAVDSS (CLINE)
      ELSE IF . . .
C
      . . .
C
C     End of reading input. At this point, any data read from the
C     DSS file would have already occurred. Close the DSS input file.
```

```
      IF (LDSSIN) CALL ZCLOSE (IFLTAB)
C
      . . .
      SUBROUTINE DSSIN (CDSSIN, CLINE, IFLTAB)
C
C     GET DATA FROM DSS
C
C     Read time series data specified on the ZR card from DSS.
C     Put flow data in array FLOWIN and precipitation data in PRECIN.
C
      CHARACTER CDSSIN*(*), CLINE*(*)
      INTEGER IFLTAB(600)
C
      CHARACTER CA*32, CC*32, CE*8, CF*32, CPATH*80, CDUM*1
      CHARACTER CLOC(10)*20, CDATE*20, CTIME*4, CTEMP*64
      INTEGER JSTATS(6)
      INTEGER*4 INTL
      REAL VALUES(500), FLOWIN(500,10), PRECIN(500,10)
      LOGICAL LFIRST, LEXIST
C     (Several variables are passed by common blocks.)
C
      DATA LFIRST /.TRUE./
      DATA CA, CC, CE, CF, CPATH /5*' '/
C
C
C     Open the DSS file the first time this subroutine is executed.
      IF (LFIRST) THEN
         LFIRST = .FALSE.
C        Make sure that the DSS input file exists.
         CTEMP = CDSSIN
         CALL ZFNAME (CTEMP, CDSSIN, NDSSIN, LEXIST)
         IF (.NOT.LEXIST) THEN
C           The DSS input file does not exist:  No data can be read in.
C           Print error message and stop.
            WRITE (6,*)'The DSS input file does not exist!', CDSSIN
            STOP
         ENDIF
C        Open the DSS file.
         CALL ZOPEN (IFLTAB, CDSSIN, ISTAT)
         IF (ISTAT.NE.0) GO TO 900
C
C        Check that a time window (from a time card) has been given.
C        Also, make sure that the number of data values to retrieve (NVALS)
C        is defined.
         . . .
C
C        Put the time interval in the E part.
```

```
            IS = 2
            CALL ZGINTL (INTL, CE, ND, IS)
            IF (IS.NE.0) GO TO 900
         ENDIF
C
C        Obtain the pathname parts from this card.  (The locations
C        names are already provided in array CLOC).
C        First, indicate not to look for the B, D, or E parts.
         JSTATS(2) = -2
         JSTATS(4) = -2
         JSTATS(5) = -2
         CALL ZGPNP (CLINE, CA, CDUM, CC, CDUM, CDUM, CF, JSTATS)
C
C        Check that we have a valid "C" part.
         IF ((CC.NE.'FLOW').AND.(CC.NE.'PRECIP')) GO TO 900
C
C        Read data from DSS for each location specified.
C
         DO 100 I=1,NLOCS
C
C           Put the pathname together.
            CALL ZPATH (CA, CLOC(I), CC, ' ', CE, CF, CPATH, NPATH)
C
C           Now retrieve the data.
            CALL ZRRTS (IFLTAB, CPATH, CDATE, CTIME, NVALS, VALUES,
        *   CUNITS, CTYPE, IOFSET, ISTAT)
C           Check for a fatal error.
            IF (ISTAT.GT.10) GO TO 940
C
C           Now transfer the data into the proper input arrays,
C           while checking for missing data (-901 or -902).  If
C           missing data is found, interpolate or set to 0.
            IF (CC(1:4).EQ.'FLOW') THEN
                CALL DATRAN (CC, VALUES, FLOWIN, NVALS, IOFSET, ISTAT)
            ELSE
                CALL DATRAN (CC, VALUES, PRECIN, NVALS, IOFSET, ISTAT)
            ENDIF
            IF (ISTAT.NE.0) GO TO 960
C
     100 CONTINUE
C
C        All done reading data for this parameter.
         RETURN
C
C
C        ERROR PROCESSING.
     900 CONTINUE
         . . .
```

```
      END
      SUBROUTINE DSSOUT (CDSSOT)
C
C     WRITE DATA OUT TO DSS
C
C     After all the data has been computed, write out the
C     computed flows and stages.
C
      CHARACTER CDSSOT*(*)
C
      CHARACTER CA*32, CE*8, CF*32, CPATH*80, CDUM*1
      CHARACTER CLINE*80, CLOC(10)*20, CDATE*9, CTIME*4
      INTEGER IFLTAB(600), JSTATS(6)
      REAL FLOOUT(500,10), STGOUT(500,10)
      LOGICAL LSTORF(10), LSTORS(10)
      (Several variables are passed by common blocks.)
C
C
C     Open the DSS file (it will be created if it does not exist).
      CALL ZOPEN (IFLTAB, CDSSOT, ISTAT)
      IF (ISTAT.NE.0) GO TO 900
C     Set the name of the program (COMFLO).
      CALL ZSET ('PROGRAM', 'COMFLO', IDUM)
C
C     Obtain the A and F pathname parts from the line saved earlier
C     by subroutine CLINE (which is passed via a common block).
C     First, indicate not to look for the B, C, D, or E parts.
      JSTATS(2) = -2
      JSTATS(3) = -2
      JSTATS(4) = -2
      JSTATS(5) = -2
      CALL ZGPNP (CLINE, CA, CDUM, CDUM, CDUM, CDUM, CF, JSTATS)
C
C     Store data in DSS for each location and parameter specified.
C
      DO 100 I=1,NLOCS
C
C        Is the computed flow to be stored?
         IF (LSTORF(I)) THEN
C
C           Put the pathname together.
            CALL ZPATH (CA, CLOC(I), 'FLOW', ' ', CE, CF, CPATH, NPATH)
C
C           Store the flow data (date and time computed earlier).
            CALL ZSRTS (IFLTAB, CPATH, CDATE, CTIME, NVALS,
     *         FLOOUT(1,I), 'CFS', 'PER-AVER', 0, ISTAT)
C           Check for a fatal error.
            IF (ISTAT.GT.10) GO TO 940
```

```
C
                ENDIF
C
C               Is the computed stage to be stored?
                IF (LSTORS(I)) THEN
C
C                   Put the pathname together.
                    CALL ZPATH (CA, CLOC(I), 'STAGE', ' ', CE, CF, CPATH, NPATH)
C
C                   Store the stage data (date and time computed earlier).
                    CALL ZSRTS (IFLTAB, CPATH, CDATE, CTIME, NVALS,
    *           STGOUT(1,I), 'FEET', 'PER-AVER', 0, ISTAT)
C                   Check for a fatal error
                    IF (ISTAT.GT.10) GO TO 940
C
                ENDIF
C
C
      100   CONTINUE
C
C           Done storing data.
            CALL ZCLOSE (IFLTAB)
            RETURN
C
C
C           Error Processing.
      900   CONTINUE
            . . .
            CALL ZCLOSE (IFLTAB)
            RETURN
            END
```

Appendix B - Internal Subroutines

The following describes the subroutines that are used internally to the DSS, and which DSS subroutines call them. These subroutines are not intended to be called directly by the programmer; they are provided for general information and as an aid in tracking down program errors.

ZABORT - Abort upon a Fatal Error

Purpose: When a fatal (un-recoverable) error occurs (such as no more disk space left), this causes an abnormal termination of the software.

Called By: ZBDUMP ZCHECK ZERROR ZGTREC
ZMULTU ZOPEN ZPTREC ZTAGPA
ZWRBUF

ZASSIG - Assign DSS File

Purpose: Makes a "shared assignment" or "exclusive assignment" for DSS files on HARRIS computers only. Creates the DSS file, if not present.

Called By: ZOPEN

ZBDUMP - Dump Buffers

Purpose: Causes all internal buffers (that have been modified) to be written to disk, and then cleared from memory. This normally occurs at the completion of every write.

Called By: ZGTREC ZMULTU ZOPEN ZRRTS
ZSRTSX

ZBEGDT - Beginning Date

Purpose: Determines the standard block start date for regular interval time-series data.

Called By: ZCAOUT ZRRTSX ZSRTSX

ZBKDAT - Block Data

Purpose: Block data for DSS. This is usually part of the ZINIT subroutine.

Called By: ZINIT

ZCAOUT - Catalog Output

Purpose:	Copies pathnames and information from the intermediate sorted file to the catalog file and produces the condensed catalog.
Called By:	ZCAT

ZCATDR - Catalog Date Reference

Purpose:	Sets the D part of a time-series pathname to search for, when a date reference is given (e.g., "D=M-2M").
Called By:	ZSETCA

ZCATFI - Catalog File

Purpose:	This physically searches the DSS file for pathnames when a new catalog is generated. This routine can also generate an internal tag-hash code table.	
Called By:	ZCAT	ZCOFIL

ZCATIT - Catalog Title

Purpose:	Writes the catalog title information.
Called By:	ZCAT

ZERROR - Fatal Error Processing

Purpose: Prints fatal error messages that are common to several subroutines.

Called By:	ZBDUMP	ZCAT	ZCATFI	ZCHECK
	ZDELET	ZDTYPE	ZOWRIT	ZPTREC
	ZRDINF	ZRETAG	ZRITSX	ZRPD
	ZRRTSX	ZRTALL	ZRTEXT	ZRTXTA
	ZSITSX	ZSPD	ZSRTSX	ZSTAGS
	ZSTEXT	ZSTXTA	ZTAGPA	ZUDALL
	ZUNDEL			

ZFSIZE - File Size

Purpose:	Determines the internal table sizes for new files.
Called By:	ZOPEN

ZGETAD - Get Address

Purpose: Gets a file address from a physical record number and word position.

Called By: ZCOFIL ZNWBIN ZOPEN ZPTREC
ZTAGFI

ZGETAG - Generate Tag

Purpose: Generates the tag for a new record.

Called By: ZNWRIT ZRTALL

ZGETCI - Get Compression Information

Purpose: Determines if a record's pathname parts match those of the default data compression scheme for new regular-interval time series records. If it does, the associated compression method and parameters are returned.

Called By: ZPRTCI ZSETCI ZSRTSX

ZGETRW - Get Record and Word

Purpose: Gets the physical file record and relative word position from a word address.

Called By: ZCOFIL ZDEBUG ZGTREC ZNWBIN
ZOPEN ZPTREC ZTAGFI

ZGTAGS - Get Tag Scheme

Purpose: Gets the default tag scheme for a file.

Called By: External programs

ZGTREC - Get a Disk Record

Purpose: Obtains a section of the DSS file. If the data requested is in memory, it is transferred and not physically read.

Called By: ZCATFI ZCHECK ZCOFIL ZCOREC
ZDELET ZGETCI ZGTAGS ZMULTU
ZNWBIN ZOPEN ZOWRIT ZPRTCI
ZRDBUF ZRDINF ZREADX ZRECIN
ZRETAG ZRRTSB ZRRTSX ZRTALL
ZSETCI ZSQPRM ZSRTSX ZTAGFI
ZTAGPA ZUDALL ZUNDEL ZUPRTS
ZWRITX

ZHASH - Determine the Hash Code

Purpose: Obtains the hash code for a given pathname and hash table size.

Called By: ZCHECK

ZINCBK - Increment Block

Purpose: Increments the dates of a time series block.

Called By: ZCAOUT ZRRTSX ZSITSX ZSRTSX

ZINIT - Initialize Variables

Purpose: Initializes the variables used by DSS. This subroutine is only called once (regardless of the number of DSS files opened).

Called By: ZINQIR ZOPEN ZSET

ZIRBEG - Irregular Beginning Date

Purpose: Determines the start date and block length for irregular-interval time series data.

Called By: ZCAOUT ZRITSX ZSITSX

ZIRDOW - Irregular Down

Purpose: Moves data in a buffer array down for insertion of additional data when storing irregular-interval time series data.

Called By: ZSITSX

ZLAHEY- Lahey® Open Adjustment

Purpose: If a Microsoft® FORTRAN program previously wrote to the file, ZLAHEY adjusts the file size so that the last record can be read by Lahey® FORTRAN.

Called By: ZOPEN

ZMATCA - Match Catalog Pathname Parts

Purpose: Matches pathname parts for the selective catalog feature.

Called By: ZSELCA

ZMIN2R - Minutes to Real

Purpose: Converts irregular-interval time series minute's array to a real array composed of Julian dates and fractions of a day.

Called By: ZGIRTS ZPIRTS

ZMOVBK - Move a Data Block

Purpose: Transfers data out of a regular-interval time series record into a data array.

Called By: ZSRTSX

ZMULTU - Multiple User

Purpose: Initializes and dumps buffers, and takes care of multiple user accesses. This subroutine is very system dependent.

Called By:

ZCATFI	ZCLOSE	ZCOFIL	ZCOREC
ZDELET	ZRENAM	ZRETAG	ZRTALL
ZSETCI	ZSETPR	ZSRTSX	ZSTAGS
ZTAGFI	ZUDALL	ZUNDEL	ZWRBUF
ZWRITX			

ZNWBIN - New Bin

Purpose: Generates a new pathname bin when needed.

Called By: ZNWRIT

ZNWRIT - New Write

Purpose: Prepares addresses, bins, space and the information block for new records. This routine is called whenever a new record is written.

Called By:

ZCOFIL	ZCOREC	ZRENAM	ZSRTSX
ZWRBUF	ZWRITX		

ZORDPN - Order Pathnames

Purpose:	Obtains pathnames from an already existing catalog file for ZCAT.
Called By:	ZCAT

ZOWRIT - Old Write

Purpose:	Updates addresses and the information block when existing records are re-written.			
Called By:	ZCOFIL	ZCOREC	ZSRTSX	ZWRBUF
	ZWRITX			

ZPRTC - Print Compression

Purpose:	Prints information about the file's default data compression settings.
Called By:	ZPRTCI

ZPTREC - Put a Disk Record

Purpose:	Stores a set of information in the DSS file. ZPTREC actually transfers data into memory. It is not written to disk unless all the internal buffers are full or until the transaction is complete.			
Called By:	ZCHECK	ZCOFIL	ZCOREC	ZDELET
	ZNWBIN	ZNWRIT	ZOPEN	ZOWRIT
	ZRENAM	ZRETAG	ZRTALL	ZSETCI
	ZSETPR	ZSQPRM	ZSRTSX	ZSTAGS
	ZTAGFI	ZUDALL	ZUNDEL	ZUPRTS
	ZWRBUF	ZWRITX		

ZR2MIN -Real to Minutes

Purpose:	Converts a real array composed of Julian dates and fractions of a day to minutes for irregular-interval time series data.
Called By:	ZPIRTS

ZRDINF - Read Information Block

Purpose:	Reads a record's information block.			
Called By:	ZRDBUF	ZRECIN	ZRRTSX	ZWRBUF
	ZWRITX			

ZRREC - Read Record

Purpose: Reads a physical record from the disk. All actual reading from the DSS file occurs in this subroutine.

Called By: ZGTREC ZPTREC

ZRRTSB - Retrieve Regular-Interval Time Series Buffer

Purpose: Reads only the portion of the disk requested for regular-interval time series data (as opposed to the entire record).

Called By: ZRRTSX

ZSELCA - Selective Catalog

Purpose: Determines which pathnames match selective catalog parameters.

Called By: ZCATFI ZORDPN

ZSETCA - Set Catalog Parameters

Purpose: Sets parameters to be used by ZSELCA.

Called By: ZCAT

ZTAGFI - Tag File

Purpose: Writes a new tag - hash code table when the file is cataloged or squeezed.

Called By: ZCATFI

ZUPRTS - Update Regular-Interval Time Series

Purpose: Writes only that portion of a record that needs to be updated when storing regular-interval time series data. (If only three values are to be written to an existing record, then only a portion of the record is written instead of the entire record).

Called By: ZSRTSX

ZWREC - Write Record

Purpose:	Writes a physical record to the disk. All actual writing to the DSS file occurs in this subroutine.
Called By:	ZBDUMP ZGTREC ZPTREC

Appendix C - Data Screening Use of Data Flags

This appendix illustrates the bit settings of data flags used by data screening software that may be stored with time-series data. Data flags written to DSS consist of thirty-two bits. The bits connote the following information about the associated data value:

Bit	**Representation**
1:	Screened - set when original data has been screened

Quality of original data:

2:	Okay
3:	Missing
4:	Questionable
5:	Reject
6-7:	Range of current data - an integer in [0,3]
8:	Current value is different from original value
9-11:	Who set current value - an integer in [0,7]:
	0 original value, no revision
	1 DATCHK program
	2 DATVUE program
	3 manual entry in the DATVUE program
	4 original value accepted in the DATVUE program
12-15:	Replacement method - an integer in [0,15]:
	0 no revision
	1 linear interpolation
	2 manual changes
	3 replace with missing value

Tests Failed:

16:	absolute magnitude
17:	constant value
18:	rate-of-change
19	relative magnitude
20:	duration-magnitude
21:	reserved for future use
22:	coefficient of variation
23:	gage list
24:	recurring value

Other:

25-31:	reserved for future use
32:	protect value from being automatically changed

Appendix D - Cross Reference Listing

The following is a cross reference list of which HECLIB subroutines are called by the DSS routines, and an inverse list showing the DSS subroutines called by the HECLIB routines. These lists are for aiding the design of a program overlay structure and for general information. The lists encompass software for several compilers. Not all the routines are necessarily called by the compiler you are using.

ZABORT calls:	ABORT	GETNAM	WAIT	WHEN
	WIND			
ZASSIG calls:	CASSIG	CCREAT	CRETYP	WAITS
ZBDUMP calls:	ZABORT	ZERROR	ZINQIR	ZWREC
ZBEGDT calls:	JLIYMD			
ZBKDAT calls:	Nothing			
ZCAOUT calls:	CHRLNB	DATJUL	ZBEGDT	ZGINTL
	ZINCBK	ZIRBEG	ZUPATH	
ZCAT calls:	CASSIG	CCREAT	CHRLNB	FILEN
	GETI1	GETNAM	system	ZCAOUT
	ZCATFI	ZCATIT	ZERROR	ZORDPN
	ZSETCA			
ZCATDR calls:	CHRLNB	CURTIM	INCTIM	INTGR
	IYMDJL	JLIYMD	JULDAT	
ZCATFI calls:	CHRWT	HOL2CH	HOLCHR	ZERROR
	ZGTREC	ZMULTU	ZSELCA	ZTAGFI
ZCATIT calls:	CHRFLB	CHRLNB	CURTIM	DATJUL
	JULDAT	M2IHM	ZINQIR	
ZCATLG calls:	ZCAT			
ZCHECK calls:	CH2HOL	HOL2CH	ZABORT	ZERROR
	ZGTREC	ZHASH	ZPTREC	
ZCHKPN calls:	Nothing			
ZCLOSE calls:	CHRLNB	CLOSF	ZINQIR	ZMULTU

ZCOFIL calls:	CHRWT	HOLCHR	ZCATFI	ZCHECK
	ZCOREC	ZGETAD	ZGETRW	ZGTREC
	ZINQIR	ZMULTU	ZNWRIT	ZOWRIT
	ZPTREC			
ZCOREC calls:	CHRLNB	DATJUL	HOLCHR	JULDAT
	M2IHM	ZCHECK	ZGINTL	ZGTREC
	ZMULTU	ZNWRIT	ZOFSET	ZOWRIT
	ZPTREC	ZREADX	ZRRTSX	ZSRTSX
	ZUPATH			
ZDCINF calls:	DHINFO			
ZDEBUG calls:	GETI1	HOL2CH	XREALC	ZGETRW
ZDELET calls:	HOL2CH	ZCHECK	ZERROR	ZGTREC
	ZMULTU	ZPTREC		
ZDTYPE calls:	CHRLNB	ZCHECK	ZERROR	ZGINTL
	ZUPATH			
ZERROR calls:	ZABORT			
ZFILST calls:	CHRFIL	CHRLNB	ZINQIR	
ZFNAME calls:	CHRFLB	UPCASE		
ZFPN calls:	ZPATH			
ZFSIZE calls:	UPCASE			
ZFVER calls:	HOLCHR	ZFNAME		
ZGETAD calls:	Nothing			
ZGETAG calls:	ISCAN	REMBLK	ZUPATH	
ZGETCI calls:	CHGTYP	CHRLNB	GETI1	HOLCHR
	ZGTREC			
ZGETRW calls:	Nothing			
ZGINTL calls:	Nothing			
ZGIRTS calls:	ZMIN2R	ZRITSX		
ZGPNP calls:	CHRLNB	PARSEQ		

ZGTAGS calls:	ZGTREC			
ZGTDTS calls:	DATCLN	JULDAT	M2IHM	NOPERS
	ZADDHD	ZFPN	ZGINTL	ZRRTSX
ZGTPFD calls:	ZRPD			
ZGTREC calls:	ZABORT	ZBDUMP	ZGETRW	ZRREC
	ZWREC			
ZHASH calls:	Nothing			
ZINCBK calls:	IYMDJL	JLIYMD		
ZINIT calls:	ABORT	WHEN	ZBKDAT	
ZINQIR calls:	GETNAM	HOL2CH	HOLCHR	ZINIT
ZINTBK calls:	IDAYWK	INCTIM	JLIYMD	NOPERS
	ZOFSET			
ZIRBEG calls:	JLIYMD			
ZIRDOW calls:	Nothing			
ZLAHEY calls:	HOLCHR			
ZMATCA calls:	Nothing			
ZMIN2R calls:	Nothing			
ZMOVBK calls:	JULDAT	LEQNER	NOPERS	
ZMULTU calls:	FLLKOF	FLLKON	LOCKF	sync
	ZABORT	ZBDUMP	ZGTREC	
ZNWBIN calls:	ZGETAD	ZGETRW	ZGTREC	ZPTREC
ZNWRIT calls:	CH2HOL	CHRHOL	GETI1	PUTI1
	ZGETAG	ZNWBIN	ZPTREC	ZUPATH
ZOFSET calls:	DATCLL	IDAYWK	INCTIM	IYMDJL
	JLIYMD			
ZOPCAT calls:	CCREAT	CHRLNB	CRETYP	INTGR
	UPCASE			

ZOPEN calls:	CH2HOL	CHRHOL	CHRLNB	CLOSF
	CREAF	FILEN	GETNAM	HOLCHR
	LFINFO	OPENF	WHEN	ZABORT
	ZASSIG	ZBDUMP	ZFNAME	ZFSIZE
	ZGETAD	ZGETRW	ZGTREC	ZINIT
	ZLAHEY	ZPTREC		
ZOPNCA calls:	CCREAT	CHRLNB	CRETYP	INTGR
	UPCASE			
ZORDPN calls:	ZRDPAT	ZSELCA		
ZOWRIT calls:	CHRHOL	HOL2CH	ZDEBUG	ZERROR
	ZGTREC	ZPTREC		
ZPATH calls:	CHRFLB			
ZPIRTS calls:	ZMIN2R	ZR2MIN	ZSITS	
ZPRTC calls:	CHGTYP	GETI1	HOLCHR	
ZPRTCI calls:	CHRLNB	ZGETCI	ZGTREC	ZPRTC
ZPTDTS calls:	DATCLN	JULDAT	M2IHM	NOPERS
	ZFPN	ZGINTL	ZSRTSX	
ZPTREC calls:	ZABORT	ZERROR	ZGETAD	ZGETRW
	ZINQIR	ZRREC	ZWREC	
ZR2MIN calls:	Nothing			
ZRDBUF calls:	CH2HOL	CHRLNB	HOL2CH	ZGTREC
	ZRDINF			
ZRDCAT calls:	CHRLNB			
ZRDINF calls:	CHRLNB	HOL2CH	ZCHECK	ZERROR
	ZGTREC	ZINQIR		
ZRDPAT calls:	CHRLNB			
ZRDPN calls:	ZRDPAT			
ZREAD calls:	CHRLNB	ZRDBUF	ZREADX	
ZREADX calls:	CHRLNB	ZGTREC	ZRDBUF	

ZRECIN calls:	CHRLNB	DHINFO	HOLCHR	ZGTREC
	ZRDINF			
ZRENAM calls:	HOLCHR	ZCHECK	ZDELET	ZMULTU
	ZNWRIT	ZPTREC		
ZRETAG calls:	CHRHOL	ZCHECK	ZERROR	ZGTREC
	ZMULTU	ZPTREC		
ZRITS calls:	ZRITSX			
ZRITSX calls:	CHRLNB	DATCLL	HOLCHR	IYMDJL
	JLIYMD	JULDAT	ZERROR	ZGIRTS
	ZINQIR	ZIRBEG	ZR2MIN	ZREADX
	ZUPATH			
ZRPD calls:	CHRLNB	HOLCHR	ZERROR	ZINQIR
	ZREADX			
ZRREC calls:	READF	SEEKF		
ZRRTS calls:	ZRRTSX			
ZRRTSB calls:	DUREAL	JULDAT	NOPERS	ZGTREC
ZRRTSX calls:	CHRLNB	DATJUL	HOLCHR	IDAYWK
	IHM2M	INCTIM	IYMDJL	JULDAT
	M2IHM	NINDX	NOPERS	YMDDAT
	ZBDUMP	ZBEGDT	ZERROR	ZGINTL
	ZGTREC	ZINCBK	ZOFSET	ZRDINF
	ZRRTS	ZRRTSB	ZUPATH	
ZRTALL calls:	CHRHOL	HOL2CH	ZERROR	ZGETAG
	ZGTREC	ZMULTU	ZPTREC	
ZRTEXT calls:	CHRLNB	HOLCHR	ZERROR	ZINQIR
	ZREADX			
ZRTXTA calls:	CHRLNB	HOLCHR	ZERROR	ZINQIR
	ZREADX			
ZSCOMP calls:	Nothing			
ZSELCA calls:	CHRLNB	DATJUL	ZGINTL	ZMATCA
	ZUPATH			
ZSET calls:	ZINIT			

ZSETCA calls:	CHRLNB	ZCATDR	ZGPNP	
ZSETCI calls:	CHGTYP	CHRHOL	CHRLNB	PUTI1
	ZGETCI	ZGTREC	ZMULTU	ZPTREC
ZSETPR calls:	ZMULTU	ZPTREC		
ZSITS calls:	ZSITSX			
ZSITSX calls:	CHRHOL	CHRLNB	DATCLL	IYMDJL
	JULDAT	M2IHM	ZERROR	ZINCBK
	ZINQIR	ZIRBEG	ZIRDOW	ZMIN2R
	ZPIRTS	ZR2MIN	ZREADX	ZSET
	ZUPATH	ZWRITX		
ZSPD calls:	CHRHOL	CHRLNB	ZERROR	ZWRITX
ZSQPRM calls:	CHRHOL	HOLCHR	ZGTREC	ZPTREC
ZSRTS calls:	ZSRTSX			
ZSRTSX calls:	CHRHOL	CHRLNB	DATJUL	DCREAL
	DUREAL	HOL2CH	IDAYWK	IHM2M
	INCTIM	IYMDJL	JULDAT	M2IHM
	NOPERS	YMDDAT	ZBDUMP	ZBEGDT
	ZCHECK	ZERROR	ZGETCI	ZGINTL
	ZGTREC	ZINCBK	ZINQIR	ZMOVBK
	ZMULTU	ZNWRIT	ZOFSET	ZOWRIT
	ZPATH	ZPTREC	ZSRTS	ZUFPN
	ZUPRTS			
ZSTAGS calls:	INTGR	PARSLI	ZERROR	ZMULTU
	ZPTREC			
ZSTEXT calls:	CHRHOL	CHRLNB	ZERROR	ZWRITX
ZSTFH calls:	CHRFLB	CHRHOL	HOLCHR	INTGR
ZSTXTA calls:	CHRHOL	CHRLNB	ZERROR	ZWRITX
ZTAGFI calls:	ZGETAD	ZGETRW	ZGTREC	ZMULTU
	ZPTREC			
ZTAGPA calls:	HOL2CH	ZABORT	ZERROR	ZGTREC
ZTSINT calls:	INCTIM	JULDAT	M2IHM	NOPERS
	ZFPN	ZGINTL	ZINTBK	ZOFSET
	ZRRTS			

ZUDALL calls:	HOL2CH	ZERROR	ZGTREC	ZMULTU
	ZPTREC			
ZUFPN calls:	CHRLNB	ZUPATH		
ZUNDEL calls:	HOL2CH	ZCHECK	ZERROR	ZGTREC
	ZMULTU	ZPTREC	ZUDALL	
ZUPATH calls:	CHRLNB			
ZUPRTS calls:	CHRHOL	JULDAT	LEQNER	NOPERS
	ZGTREC	ZPTREC		
ZUSTFH calls:	CHRLNB	GETHOL	HOLCHR	INTGR
ZWRBUF calls:	CH2HOL	CHRLNB	HOL2CH	ZABORT
	ZCHECK	ZMULTU	ZNWRIT	ZOWRIT
	ZPTREC	ZRDINF		
ZWREC calls:	SEEKF	WRITF		
ZWRITE calls:	ZWRITX			
ZWRITX calls:	ZCHECK	ZGTREC	ZMULTU	ZNWRIT
	ZOWRIT	ZPTREC	ZRDINF	

ABORT is called by:	ZABORT	ZINIT		
CASSIG is called by:	ZCAT			
CCREAT is called by:	ZASSIG	ZCAT	ZOPCAT	ZOPNCA
CH2HOL is called by:	ZCHECK	ZNWRIT	ZOPEN	ZRDBUF
	ZWRBUF			
CHGTYP is called by:	ZGETCI	ZPRTC	ZSETCI	
CHRFIL is called by:	ZFILST			
CHRFLB is called by:	ZCATIT	ZFNAME	ZPATH	ZSTFH
CHRHOL is called by:	ZNWRIT	ZOPEN	ZOWRIT	ZRETAG
	ZRTALL	ZSETCI	ZSITSX	ZSPD
	ZSQPRM	ZSRTSX	ZSTEXT	ZSTFH
	ZSTXTA	ZUPRTS		

CHRLNB is called by:	ZCAOUT	ZCAT	ZCATDR	ZCATIT
	ZCLOSE	ZCOREC	ZDTYPE	ZFILST
	ZGETCI	ZGPNP	ZOPCAT	ZOPEN
	ZOPNCA	ZPRTCI	ZRDBUF	ZRDCAT
	ZRDINF	ZRDPAT	ZREAD	ZREADX
	ZRECIN	ZRITSX	ZRPD	ZRRTSX
	ZRTEXT	ZRTXTA	ZSELCA	ZSETCA
	ZSETCI	ZSITSX	ZSPD	ZSRTSX
	ZSTEXT	ZSTXTA	ZUFPN	ZUPATH
	ZUSTFH	ZWRBUF		
CHRWT is called by:	ZCATFI	ZCOFIL		
CLOSF is called by:	ZCLOSE	ZOPEN		
CREAF is called by:	ZOPEN			
CRETYP is called by:	ZASSIG	ZOPCAT	ZOPNCA	
CURTIM is called by:	ZCATDR	ZCATIT		
DATCLL is called by:	ZOFSET	ZRITSX	ZSITSX	
DATCLN is called by:	ZGTDTS	ZPTDTS		
DATJUL is called by:	ZCAOUT	ZCATIT	ZCOREC	ZRRTSX
	ZSELCA	ZSRTSX		
DCREAL is called by:	ZSRTSX			
DHINFO is called by:	ZDCINF	ZRECIN		
DUREAL is called by:	ZRRTSB	ZSRTSX		
FILEN is called by:	ZCAT	ZOPEN		
FLLKOF is called by:	ZMULTU			
FLLKON is called by:	ZMULTU			
GETHOL is called by:	ZUSTFH			
GETI1 is called by:	ZCAT	ZDEBUG	ZGETCI	ZNWRIT
	ZPRTC			
GETNAM is called by:	ZABORT	ZCAT	ZINQIR	ZOPEN

HOL2CH is called by:	ZCATFI	ZCHECK	ZDEBUG	ZDELET
	ZINQIR	ZOWRIT	ZRDBUF	ZRDINF
	ZRTALL	ZSRTSX	ZTAGPA	ZUDALL
	ZUNDEL	ZWRBUF		
HOLCHR is called by:	ZCATFI	ZCOFIL	ZCOREC	ZFVER
	ZGETCI	ZINQIR	ZLAHEY	ZOPEN
	ZPRTC	ZRECIN	ZRENAM	ZRITSX
	ZRPD	ZRRTSX	ZRTEXT	ZRTXTA
	ZSQPRM	ZSTFH	ZUSTFH	
IDAYWK is called by:	ZINTBK	ZOFSET	ZRRTSX	ZSRTSX
IHM2M is called by:	ZRRTSX	ZSRTSX		
INCTIM is called by:	ZCATDR	ZINTBK	ZOFSET	ZRRTSX
	ZSRTSX	ZTSINT		
INTGR is called by:	ZCATDR	ZOPCAT	ZOPNCA	ZSTAGS
	ZSTFH	ZUSTFH		
ISCAN is called by:	ZGETAG			
IYMDJL is called by:	ZCATDR	ZINCBK	ZOFSET	ZRITSX
	ZRRTSX	ZSITSX	ZSRTSX	
JLIYMD is called by:	ZBEGDT	ZCATDR	ZINCBK	ZINTBK
	ZIRBEG	ZOFSET	ZRITSX	
JULDAT is called by:	ZCATDR	ZCATIT	ZCOREC	ZGTDTS
	ZMOVBK	ZPTDTS	ZRITSX	ZRRTSB
	ZRRTSX	ZSITSX	ZSRTSX	ZTSINT
	ZUPRTS			
LEQNER is called by:	ZMOVBK	ZUPRTS		
LFINFO is called by:	ZOPEN			
LOCKF is called by:	ZMULTU			
M2IHM is called by:	ZCATIT	ZCOREC	ZGTDTS	ZPTDTS
	ZRRTSX	ZSITSX	ZSRTSX	ZTSINT
NINDX is called by:	ZRRTSX			
NOPERS is called by:	ZGTDTS	ZINTBK	ZMOVBK	ZPTDTS
	ZRRTSB	ZRRTSX	ZSRTSX	ZTSINT
	ZUPRTS			

OPENF is called by:	ZOPEN			
PARSEQ is called by:	ZGPNP			
PARSLI is called by:	ZSTAGS			
PUTI1 is called by:	ZNWRIT	ZSETCI		
READF is called by:	ZRREC			
REMBLK is called by:	ZGETAG			
SEEKF is called by:	ZRREC	ZWREC		
sync is called by:	ZMULTU			
system is called by:	ZCAT			
UPCASE is called by:	ZFNAME	ZFSIZE	ZOPCAT	ZOPNCA
WAIT is called by:	ZABORT			
WAITS is called by:	ZASSIG			
WHEN is called by:	ZABORT	ZINIT	ZOPEN	
WIND is called by:	ZABORT			
WRITF is called by:	ZWREC			
XREALC is called by:	ZDEBUG			
YMDDAT is called by:	ZRRTSX	ZSRTSX		
ZABORT is called by:	ZBDUMP	ZCHECK	ZERROR	ZGTREC
	ZMULTU	ZOPEN	ZPTREC	ZTAGPA
	ZWRBUF			
ZADDHD is called by:	ZGTDTS			
ZASSIG is called by:	ZOPEN			
ZBDUMP is called by:	ZGTREC	ZMULTU	ZOPEN	ZRRTSX
	ZSRTSX			
ZBEGDT is called by:	ZCAOUT	ZRRTSX	ZSRTSX	

ZBKDAT is called by:	ZINIT			
ZCAOUT is called by:	ZCAT			
ZCAT is called by:	ZCATLG			
ZCATDR is called by:	ZSETCA			
ZCATFI is called by:	ZCAT	ZCOFIL		
ZCATIT is called by:	ZCAT			
ZCHECK is called by:	ZCOFIL	ZCOREC	ZDELET	ZDTYPE
	ZRDINF	ZRENAM	ZRETAG	ZSRTSX
	ZUNDEL	ZWRBUF	ZWRITX	
ZCOREC is called by:	ZCOFIL			
ZDEBUG is called by:	ZOWRIT			
ZDELET is called by:	ZRENAM			
ZERROR is called by:	ZBDUMP	ZCAT	ZCATFI	ZCHECK
	ZDELET	ZDTYPE	ZOWRIT	ZPTREC
	ZRDINF	ZRETAG	ZRITSX	ZRPD
	ZRRTSX	ZRTALL	ZRTEXT	ZRTXTA
	ZSITSX	ZSPD	ZSRTSX	ZSTAGS
	ZSTEXT	ZSTXTA	ZTAGPA	ZUDALL
	ZUNDEL			
ZFNAME is called by:	ZFVER	ZOPEN		
ZFPN is called by:	ZGTDTS	ZPTDTS	ZTSINT	
ZFSIZE is called by:	ZOPEN			
ZGETAD is called by:	ZCOFIL	ZNWBIN	ZOPEN	ZPTREC
	ZTAGFI			
ZGETAG is called by:	ZNWRIT	ZRTALL		
ZGETCI is called by:	ZPRTCI	ZSETCI	ZSRTSX	
ZGETRW is called by:	ZCOFIL	ZDEBUG	ZGTREC	ZNWBIN
	ZOPEN	ZPTREC	ZTAGFI	

ZGINTL is called by:	ZCAOUT	ZCOREC	ZDTYPE	ZGTDTS
	ZPTDTS	ZRRTSX	ZSELCA	ZSRTSX
	ZTSINT			
ZGIRTS is called by:	ZRITSX			
ZGPNP is called by:	ZSETCA			
ZGTREC is called by:	ZCATFI	ZCHECK	ZCOFIL	ZCOREC
	ZDELET	ZGETCI	ZGTAGS	ZMULTU
	ZNWBIN	ZOPEN	ZOWRIT	ZPRTCI
	ZRDBUF	ZRDINF	ZREADX	ZRECIN
	ZRETAG	ZRRTSB	ZRRTSX	ZRTALL
	ZSETCI	ZSQPRM	ZSRTSX	ZTAGFI
	ZTAGPA	ZUDALL	ZUNDEL	ZUPRTS
	ZWRITX			
ZHASH is called by:	ZCHECK			
ZINCBK is called by:	ZCAOUT	ZRRTSX	ZSITSX	ZSRTSX
ZINIT is called by:	ZINQIR	ZOPEN	ZSET	
ZINQIR is called by:	ZBDUMP	ZCATIT	ZCLOSE	ZCOFIL
	ZFILST	ZPTREC	ZRDINF	ZRITSX
	ZRPD	ZRTEXT	ZRTXTA	ZSITSX
	ZSRTSX			
ZINTBK is called by:	ZTSINT			
ZIRBEG is called by:	ZCAOUT	ZRITSX	ZSITSX	
ZIRDOW is called by:	ZSITSX			
ZLAHEY is called by:	ZOPEN			
ZMATCA is called by:	ZSELCA			
ZMIN2R is called by:	ZGIRTS	ZPIRTS	ZSITSX	
ZMOVBK is called by:	ZSRTSX			
ZMULTU is called by:	ZCATFI	ZCLOSE	ZCOFIL	ZCOREC
	ZDELET	ZRENAM	ZRETAG	ZRTALL
	ZSETCI	ZSETPR	ZSRTSX	ZSTAGS
	ZTAGFI	ZUDALL	ZUNDEL	ZWRBUF
	ZWRITX			

ZNWBIN is called by:	ZNWRIT			
ZNWRIT is called by:	ZCOFIL	ZCOREC	ZRENAM	ZSRTSX
	ZWRBUF	ZWRITX		
ZOFSET is called by:	ZCOREC	ZINTBK	ZRRTSX	ZSRTSX
	ZTSINT			
ZORDPN is called by:	ZCAT			
ZOWRIT is called by:	ZCOFIL	ZCOREC	ZSRTSX	ZWRBUF
	ZWRITX			
ZPATH is called by:	ZFPN	ZSRTSX		
ZPIRTS is called by:	ZSITSX			
ZPRTC is called by:	ZPRTCI			
ZPTREC is called by:	ZCHECK	ZCOFIL	ZCOREC	ZDELET
	ZNWBIN	ZNWRIT	ZOPEN	ZOWRIT
	ZRENAM	ZRETAG	ZRTALL	ZSETCI
	ZSETPR	ZSQPRM	ZSRTSX	ZSTAGS
	ZTAGFI	ZUDALL	ZUNDEL	ZUPRTS
	ZWRBUF	ZWRITX		
ZR2MIN is called by:	ZPIRTS	ZRITSX	ZSITSX	
ZRDBUF is called by:	ZREAD	ZREADX		
ZRDINF is called by:	ZRDBUF	ZRECIN	ZRRTSX	ZWRBUF
	ZWRITX			
ZRDPAT is called by:	ZORDPN	ZRDPN		
ZREADX is called by:	ZCOREC	ZREAD	ZRITSX	ZRPD
	ZRTEXT	ZRTXTA	ZSITSX	
ZRITSX is called by:	ZGIRTS	ZRITS		
ZRPD is called by:	ZGTPFD			
ZRREC is called by:	ZGTREC	ZPTREC		
ZRRTS is called by:	ZRRTSX	ZTSINT		
ZRRTSB is called by:	ZRRTSX			

ZRRTSX is called by:	ZCOREC	ZGTDTS	ZRRTS	
ZSELCA is called by:	ZCATFI	ZORDPN		
ZSET is called by:	ZSITSX			
ZSETCA is called by:	ZCAT			
ZSITS is called by:	ZPIRTS			
ZSITSX is called by:	ZSITS			
ZSPD is called by:	ZPTPFD			
ZSRTS is called by:	ZSRTSX			
ZSRTSX is called by:	ZCOREC	ZPTDTS	ZSRTS	
ZTAGFI is called by:	ZCATFI			
ZUDALL is called by:	ZUNDEL			
ZUFPN is called by:	ZSRTSX			
ZUPATH is called by:	ZCAOUT	ZCOREC	ZDTYPW	ZGETAG
	ZNWRIT	ZRITSX	ZRRTSX	ZSELCA
	ZSITSX	ZUFPN		
ZUPRTS is called by:	ZSRTSX			
ZWREC is called by:	ZBDUMP	ZGTREC	ZPTREC	
ZWRITX is called by:	ZSITSX	ZSPD	ZSTEXT	ZSTXTA
	ZWRITE			

Appendix E – Abort Error Codes

The following is a list of error codes that may be printed if a fatal error is detected by the basic DSS software that stores or retrieves information. If one these errors occur, a message will be printed with the error, then the program will be aborted. These "low-level" errors occur relatively infrequently, and are not the same as the errors for the "high-level" subroutines (such as ZSRTS). Refer to the individual subroutine documentation for error codes for high-level subroutines.

"Keys" within several DSS arrays (such as the IFLTAB) are checked frequently to ensure that the arrays are not corrupted (by the calling program overwriting memory), and damaging the database. However, it is possible for an area of a DSS array to be corrupted and stored on disk, damaging the database file, before being detected.

Internal addresses are physically stored after the data has been stored, so that if a crash occurs (e.g., a power failure or the disk space is exceeded), only that record will be lost and the database will not be damaged. However, occasionally all internal DSS buffers may become used, and some buffers will be written prior to the data block. If this occurs at the time of a crash, the file could become damaged. Also, some computers may buffer I/O in a way that records are not physically written to disk the way that the DSS software expects (such as disk cashing). This also could cause damage to a database file if a crash occurs.

A damaged file can usually be recovered by squeezing the file with DSSUTL. (Be sure there is sufficient disk space to do so.) The cause of the error should be determined before proceeding. (It should be noted that a file with only minor damage may not be detected for a while.)

Code	**Error Description**
11	**Pointer or address array incorrect.** The address from the pathname address array (pathname bin) points to a record information block where the pathname does not match the bin pathname. This typically indicates that the area read is not a record information block, and the file is damaged.
20	**Illegal Unit number.** An invalid unit number was encountered during a write or read operation.
30	**Error on Physical Read.** An error occurred during a read operation. This may be caused by a damaged file with an address that points to a location beyond the file's bounds. Typically, the error occurs when the end-of-file is reached.
40	**Error on Physical Write.** An error occurred during a write operation. This can occur if the disk space is exceeded. This usually does not indicate a damaged file (although squeezing the file is a good idea).
41	**Disk Space Exceeded.** There is insufficient disk space left for this file to continue writing.
50	**Corrupt IFLTAB array (Keys don't match).** The IFLTAB array has been overwritten by the calling program. Check your program for a bounds error that can cause this.

Code	Error Description
60	**Incomplete Buffered write.** A buffered write using ZWRBUF was not terminated with LEND set to .TRUE..
70	**DSS File Not Opened.** A DSS subroutine was called with an invalid IFLTAB array (containing only zeros). This error usually occurs when testing modifications to a program. Usually the routine that opens the DSS file was not called, or the IFLTAB array was not passed to the routine calling DSS (e.g., it is not in common), or there is a typographical error in the code (e.g., IFLTAB given the wrong name). Less frequently this error may occur if the IFLTAB array is overwritten by the calling program.
100	**Illegal KTABLE variable (IFLTAB Corrupt).** The IFLTAB variable containing the table type did not contain a valid type. This can only occur if IFLTAB is overwritten. Check your program for a bounds error.
110	**Illegal Number of Characters per Machine word Set.** The DSS software was not installed correctly, or a common block was destroyed.
130	**Excess writes on a read only file.** The file was given "read only" permission, and excessive writes were attempted. After 40 low-level write attempts the program is aborted.
200	**Unable to make shared assignment.** A file lock was attempted, but was denied because the file did not have shared access (although it was supposedly opened with shared access).
210	**Unable to lock file.** A lock on the file was denied (nor could it que for a lock).
220	**Unable to unlock file.** The file could not be unlocked. Memory is probably overwritten for this error to occur.
300	**Incompatible Versions.** The DSS file is not a Version 6 file, and the routine accessed is Version 6 only. If you are sure the file is Version 6, and was correctly opened, then the IFLTAB array may be invalid or corrupt.
320	**Insufficient Header Space in ZWRBUF.** The total amount of user header data to store is greater than the space allocated in the first call to ZWRBUF.
330	**Insufficient Data Space in ZWRBUF.** The total number of data values to store is greater than the space allocated in the first call to ZWRBUF.

Appendix F - Summary of Subroutine Calling Sequences

ZCAT Catalog a DSS File 7-34

```
      CALL  ZCAT (IFLTAB, ICUNIT, ICDUNT, INUNIT, CINSTR,
    * LABREV, LSORT, LCDCAT, NRECS)
```

ZCATLG Catalog a DSS file 11-34

```
      CALL ZCATLG (IFLTAB, ICUNIT, INUNIT, CINSTR, IBEG,
    * NINSTR, LABREV, LSORT, NRECS)
```

ZCHECK Check if a Record Exists 9-10

```
      CALL ZCHECK (IFLTAB, CPATH, NPATH, NHEAD, NDATA,
    * LFOUND)
```

ZCHKPN Check a Pathname 3-12

```
      CALL ZCHKPN (CPATH, NPATH, ISTAT)
```

ZCLOSE Close a DSS File 2-5

```
      CALL ZCLOSE (IFLTAB)
```

ZCOFIL Copy a DSS File 9--18

```
      CALL ZCOFIL (IFTOLD, IFTNEW, BUFF1, KBUFF1, BUFF2, KBUFF2,
    * LUNDEL, LRETAG)
```

ZCOREC Copy a Record 9-13

```
      CALL ZCOREC (IFTOLD, IFTNEW, CPOLD, CPNEW, BUFF1,
    * KBUFF1, BUFF2, KBUFF2, ISTAT)
```

ZDCINF Get Data Compression Information for a Record 10-5

```
      CALL ZDCINF (ICOMP, BASEV, LBASEV, ISIZE, IPREC, ISTAT)
```

ZDEBUG Print Coded Information in the IFLTAB Array 9-24

```
      CALL ZDEBUG (MUNIT, IARRAY, IADD, ILEN)
```

ZDELET Delete a Record 9-21

```
      CALL ZDELET (IFLTAB, CPATH, NPATH, LFOUND)
```

ZDTYPE Determine a Record's Data Type and if it Exists 2-8

```
      CALL ZDTYPE (IFLTAB, CPATH, NSIZE, LEXIST, CDTYPE,
    * IDTYPE)
```

ZFILST Display Information About a DSS File 9-14

```
      CALL ZFILST (IFLTAB)
```

ZFNAME Add File Name Extension and Determine if it Exists 2-6

```
      CALL ZFNAME (CNAMIN, CNAME, NNAME, LEXIST)
```

ZFPN Form DSS Pathname 11-2

```
     CALL ZFPN (CA, NA, CB, NB, CC, NC, CD, ND, CE, NE,
*    CF, NF, CPATH, NPATH)
```

ZFVER Get a DSS File's Version 2-20

```
     CALL ZFVER (CNAME, CVER, IVER)
```

ZGINTL Get Time Series Interval 4-40

```
     CALL ZGINTL (INTL, CE, NVALS, ISTAT)
```

ZGIRTS Get Irregular-Interval Time Series Data 11-12

```
     CALL ZGIRTS (IFLTAB, CPATH, NPATH, JULS, ISTIME, JULE,
*    IETIME, DUM1, IDUM1, DUM2, IDUM2, KVALS, DATES,
*    VALUES, NVALS, BDATE, CUNITS, CTYPE, ISTAT)
```

ZGPNP Get Pathname Parts 3-8

```
     CALL ZGPNP (CLINE, CA, CB, CC, CD, CE, CF, NPARTS)
```

ZGTDTS Get Regular-Interval Time Series Data 11-4

```
     CALL ZGTDTS (IFLTAB, CA, NA, CB, NB, CC, NC, CF, NF,
*    JULS, ISTIME, JULE, IETIME, INTL, DUM, 0,
*    IOFSET, VALUES, NVALS, CUNITS, CTYPE, ISTAT)
```

ZGTPFD Get Paired Function Data 11-21

```
     CALL ZGTPFD (IFLTAB, CPATH, NPATH, NORD, NCURVE,
*    IHORIZ C1UNIT, C2UNIT, C1TYPE, C2TYPE, CLABEL, KLABEL,
*    NLABEL, DUM, IDUM1, IDUM2, VALUES, KVALS, NVALS, ISTAT)
```

ZINQIR Inquire About DSS Parameters 2-17

```
     CALL ZINQIR (IFLTAB, CITEM, CSTR, INUMB)
```

ZOFSET Determine the Time Offset of Time Series Data 4-44

```
     CALL ZOFSET (JUL, ITIME, INTL, IFLAG, IOFSET)
```

ZOPCAT -Open a Catalog File 11-32

```
     CALL ZOPCAT (CDSSFI, CATFIL, ICUNIT, LOPEN, LCATLG,
*    LCREAT, NRECS)
```

ZOPEN Open a DSS File 2-2

```
     CALL ZOPEN (IFLTAB, CNAME, IOSTAT)
```

ZOPNCA Open a Catalog File 7-3

```
     CALL ZOPNCA (CDSSFI, ICUNIT, LGENCA, LOPNCA, LCATLG,
*    ICDUNT, LGENCD, LOPNCD, LCATCD, NRECS)
```

ZPATH Construct a Pathname 3-2

```
     CALL ZPATH (CA, CB, CC, CD, CE, CF, CPATH, NPATH)
```

ZPIRTS Put Irregular-Interval Time Series Data 11-17
```
      CALL ZPIRTS (IFLTAB, CPATH, NPATH, DUM1, IDUM1, DUM2,
   *  IDUM2, DATES, VALUES, NVALS, BDATE, CUNITS, CTYPE,
   *  INFLAG, ISTAT)
```

ZPRTCI Print the Default Data Compression for a DSS file 10-10
```
      CALL ZPRTCI (IFLTAB, LALL, CPARTS)
```

ZPTDTS Put Regular-Interval Time Series Data in a DSS File 11-8
```
      CALL ZPTDTS (IFLTAB, CA, NA, CB, NB, CC, NC, CF, NF,
   *  JULS, ISTIME, JULE, IETIME, INTL, DUM1, IDUM,
   *  DUM2, VALUES, NVALS, CUNITS, CTYPE, ISTAT)
```

ZPTPFD Put Paired Function Data 11-26
```
      CALL ZPTPFD (IFLTAB, CPATH, NPATH, NORD, NCURVE, IHORIZ,
   *  C1UNIT, C2UNIT, C1TYPE, C2TYPE, CLABEL, KLABEL, NLABEL,
   *  DUM, IDUM1, IDUM2, VALUES, KVALS, NVALS, IPLAN, ISTAT)
```

ZRDBUF Read an Individual Record in a Buffered Mode 8-7
```
      CALL ZRDBUF (IFLTAB, CPATH, HEADU, KHEADU, NHEADU,
   *  DATA, KDATA, NDATA, LEND, IPLAN, LFOUND)
```

ZRDCAT Read Pathnames from a Catalog File 7-10
```
      CALL ZRDCAT (ICUNIT, LALL, IOUNIT, CTAGS, NDIM,
   *  CPATHS, NPATHS, NFOUND)
```

ZRDPAT Read a Pathname from a Catalog by Reference Number 7-14
```
      CALL ZRDPAT (ICUNIT, IPOS, INUMB, CTAG, CPATH, NPATH,
   *  LEND)
```

ZRDPN Read Pathnames from a Catalog File 11-37
```
      CALL ZRDPN (ICUNIT, IPOS, INUMB, CPATH, NPATH)
```

ZREAD Read an Individual Record 8-2
```
      CALL ZREAD (IFLTAB, CPATH, NPATH, IHEADU, NHEADU,
   *  IDATA, NDATA, IPLAN, LFOUND)
```

ZREADX Read an Individual Record (Extended) 8-5
```
      CALL ZREADX (IFLTAB, CPATH, HEADI, KHEADI, NHEADI,
   *  HEADC, KHEADC, NHEADC, HEADU, KHEADU, NHEADU, DATA,
   *  KDATA, NDATA, IPLAN, LFOUND)
```

ZRECIN Display Information About a Record 9-12
```
      CALL ZRECIN (IFLTAB, IUNIT, MLEVEL, CPATH, BUFF, KBUFF,
   *  LFOUND)
```

ZRENAM Rename a Record...9-20

```
     CALL ZRENAM (IFLTAB, CPATHO, NPATHO, CPATHN, NPATHN,
*    LFOUND)
```

ZRETAG Change a Record Tag 7-21

```
     CALL ZTAGPA (IFLTAB, CPATH, NPATH, CTAG, LFOUND)
```

ZRITS Retrieve Irregular-Interval Time Series Data 4-22

```
     CALL ZRITS (IFLTAB, CPATH, JULS, ISTIME, JULE, IETIME,
*    ITIMES, VALUES, KVALS, NVALS, JBDATE, CUNITS, CTYPE
*    , ISTAT)
```

ZRITSX Retrieve Irregular-Interval Time Series Data (Extended)....................................4-26

```
     CALL ZRITSX (IFLTAB, CPATH, JULS, ISTIME, JULE, IETIME,
*    ITIMES, VALUES, KVALS, NVALS, JBDATE, FLAGS, LFLAGS,
*    LFREAD, CUNITS, CTYPE, HEADU, KHEADU, NHEADU, INFLAG,
*    ISTAT)
```

ZRPD Retrieve Paired Data...5-2

```
     CALL ZRPD (IFLTAB, CPATH, NORD, NCURVE, IHORIZ,
*    C1UNIT, C1TYPE, C2UNIT, C2TYPE, VALUES, KVALS, NVALS,
*    CLABEL, KLABEL, LABEL, HEADU, KHEADU, NHEADU, ISTAT)
```

ZRRTS Retrieve Regular-Interval Time Series Data 4-3

```
     CALL ZRRTS (IFLTAB, CPATH, CDATE, CTIME, NVALS, VALUES,
*    CUNITS, CTYPE, IOFSET, ISTAT)
```

ZRRTSX Retrieve Regular-Interval Time Series Data (Extended) 4-8

```
     CALL ZRRTSX (IFLTAB, CPATH, CDATE, CTIME, NVALS, VALUES,
*    FLAGS, LFLAGS, LFREAD, CUNITS, CTYPE, HEADU, KHEADU,
*    NHEADU, IOFSET, ICOMP, ISTAT)
```

ZRTALL - Change all Record Tags in a DSS file 7-24

```
     CALL ZRTALL (IFLTAB)
```

ZRTEXT Retrieve Text Data (Into a File)...6-2

```
     CALL ZRTEXT (IFLTAB, CPATH, IUNIT, HEADU, KHEADU, NHEADU,
*    LCCNTL, NLINES, ISTAT)
```

ZRTXTA Retrieve Text Data (Into an Array)..6-5

```
     CALL ZRTXTA (IFLTAB, CPATH, CARRAY, KLINES, NLINES,
*    HEADU, KHEADU, NHEADU, ISTAT)
```

ZSCOMP Set Data Compression for a Record..10-3

```
     CALL ZSCOMP (ICOMP, BASEV, LBASEV, LHIGH, IPREC)
```

ZSET Set DSS Parameters....2-11

```
      CALL ZSET (CITEM, CSTR, INUMB)
```

ZSETCI Set Default Data Compression for a DSS File 10-7

```
      CALL ZSETCI (IFLTAB, CPARTS, LPARTS, ICOMP, BASEV,
   *  LBASEV, LHIGH, IPREC, ISTAT)
```

ZSITS Store Irregular-Interval Time Series Data 4-32

```
      CALL ZSITS (IFLTAB, CPATH, ITIMES, VALUES, NVALS, JBDATE,
   *  CUNITS, CTYPE, INFLAG, ISTAT)
```

ZSITSX Store Irregular-Interval Time Series Data (Extended) 4-37

```
      CALL ZSITSX (IFLTAB, CPATH, ITIMES, VALUES, NVALS,
   *  JBDATE, FLAGS, LFLAGS, CUNITS, CTYPE, HEADU, NHEADU,
   *  INFLAG, ISTAT)
```

ZSPD Store Paired Data....5-7

```
      CALL ZSPD (IFLTAB, CPATH, NORD, NCURVE, IHORIZ,
   *  C1UNIT, C1TYPE, C2UNIT, C2TYPE, VALUES,
   *  CLABEL, LABEL, HEADU, NHEADU, IPLAN, ISTAT)
```

ZSRTS Store Regular-Interval Time Series Data....4-14

```
      CALL ZSRTS (IFLTAB, CPATH, CDATE, CTIME, NVALS, VALUES,
   *  CUNITS, CTYPE, IPLAN, ISTAT)
```

ZSRTSX Store Regular-Interval Time Series Data (Extended) 4-17

```
      CALL ZSRTSX (IFLTAB, CPATH, CDATE, CTIME, NVALS, VALUES,
   *  FLAGS, LFLAGS, CUNITS, CTYPE, HEADU, NHEADU, IPLAN,
   *  ICOMP, BASEV, LBASEV, LHIGH, IPREC, ISTAT)
```

ZSTAGS Set the Tag Scheme for a DSS file....7-22

```
      CALL ZSTAGS (IFLTAB, CSCHEM, ISTAT)
```

ZSTEXT - Store Text Data (From a File)....6-8

```
      CALL ZSTEXT (IFLTAB, CPATH, IUNIT, HEADU, NHEADU,
   *  NLINES, ISTAT)
```

ZSTFH Stuff the User Header Array 9-2

```
      CALL ZSTFH (CLABEL, CITEM, NITEM, HEADU, KHEADU,
   *  NHEADU, ISTAT)
```

ZSTXTA Store Text Data (From an Array)....6-11

```
      CALL ZSTXTA (IFLTAB, CPATH, CARRAY, NLINES, HEADU,
   *  NHEADU, ISTAT)
```

ZTAGPA Get Pathnames from Tags 7-17

```
      CALL ZTAGPA (IFLTAB, IOUNIT, CTAGS, NDIM, CPATHS,
   *  NPATHS , NFOUND)
```

ZUDALL Undelete All Records in a DSS File 9-23

```
      CALL ZUDALL (IFLTAB, IUNIT)
```

ZUFPN Unform a Pathname 3-6

```
      CALL ZUFPN (CA, NA, CB, NB, CC, NC, CD, ND, CE, NE,
 *    CF, NF, CPATH, NPATH, ISTAT)
```

ZUNDEL Undelete a Record 9-22

```
      CALL ZUNDEL (IFLTAB, CPATH, NPATH, ISTAT)
```

ZUPATH Determine a Pathname's Parts 3-4

```
      CALL ZUPATH (CPATH, IBPART, IEPART, ILPART, ISTAT)
```

ZUSTFH Unstuff the User Header Array 9-6

```
      CALL ZUSTFH (CLABEL, CITEM, NITEM, IPOS, HEADU, NHEADU,
 *    ISTAT)
```

ZWRBUF Write an Individual Record in a Buffered Mode 8-16

```
      CALL ZWRBUF (IFLTAB, CPATH, HEADU, NHEADU, NTOTH,
 *    DATA, NDATA, NTOTD, LEND)
```

ZWRITE Write an Individual Record 8-10

```
      CALL ZWRITE (IFLTAB, CPATH, NPATH, IHEADU, NHEADU,
 *    IDATA, NDATA, IPLAN, LFOUND)
```

ZWRITX Write an Individual Record (Extended) 8-13

```
      CALL ZWRITX (IFLTAB, CPATH, NPATH, HEADI, NHEADI,
 *    HEADC, NHEADC, HEADU, NHEADU, DATA, NDATA, ITYPE,
 *    IPLAN, ISTAT, LFOUND)
```