CRAY RESEARCH, INC.

CRAY X-MP AND CRAY-1®
COMPUTER SYSTEMS

CRAY-OS VERSION 1
REFERENCE MANUAL
SR-0011

RECORD OF REVISION CRAY RESEARCH, INC. PUBLICATION NUMBER SR-0011

Each time this manual is revised and reprinted, all changes issued against the previous version in the form of change packets are incorporated into the new version and the new version is assigned an alphabetic level. Between reprints, changes may be issued against the current version in the form of change packets. Each change packet is assigned a numeric designator, starting with 01 for the first change packet of each revision level.

Every page changed by a reprint or by a change packet has the revision level and change packet number in the lower righthand corner. Changes to part of a page are noted by a change bar along the margin of the page. A change bar in the margin opposite the page number indicates that the entire page is new; a dot in the same place indicates that information has been moved from one page to another, but has not otherwise changed.

Requests for copies of Cray Research, Inc. publications and comments about these publications should be directed to:

CRAY RESEARCH, INC.,
1440 Northland Drive,
Mendota Heights, Minnesota 55120

Revision	Description
	June, 1976 - First printing.
A	September, 1976 - General technical changes; changes to JOB, MODE, RFL, and DMP statements; names of DS and RETURN changed to ASSIGN and RELEASE. STAGEI deleted, STAGEO replaced by DISPOSE. RECALL macro added and expansions provided for all logical I/O macros. RELEASE, DUMPDS, and LOADPDS renamed to DELETE, PDSDUMP, and PDSLOAD. Detailed description of BUILD added (formerly LIB). EDIT renamed to UPDATE.
B	February, 1977 - Addition of Overlay Loader; deletion of Loader Tables (information now documented in CRI publication SR-0012); deletion of UPDATE (information now documented in CRI publication SR-0013); changes to reflect current implementation.
C	July, 1977 - Addition of BKSPF, GETPOS, and POSITION logical I/O macros and $BKSPF, $GPOS, and $SPOS routines. Addition of random I/O. Changes to dataset structure, JOB, ASSIGN, MODE, and DUMP statements; BUILD; logical I/O and system action macro expansions. General technical changes to reflect current implementation.
C-01	January, 1978 - Correction to DISPOSE and LDR control statement documentation, addition of description of $WWDS write routine, miscellaneous changes to bring documentation into agreement with January 1978 released version of the operating system.
D	February, 1978 - Reprint with revision. This printing is exactly the same as revision C with the C-01 change packet added.
D-01	April, 1978 - Change packet includes the addition of the ADJUST control statement; MODE and SWITCH macros; and PDD, ACCESS, SAVE DELETE, and ADJUST permanent dataset macros. Miscellaneous changes to bring documentation into agreement with released system, version 1.01.

Revision	Description
E	July, 1978 - Represents a complete rewrite of this manual. Changes are not marked by change bars. New features for version 1.02 of the operating system that are documented in this revision include: addition of the MODIFY control statement and the DSP, SYSID, and DISPOSE macros; the addition of parameters to some control statements, the implementation of BUILD. The POSITION macro has been renamed SETPOS. Other changes to bring documentation into agreement with released version 1.02 of the operating system.
E-01	October, 1978 - Change packet includes the implementation of ACQUIRE and COMPARE control statements; changes to the AUDIT and LDR control statements; changes to the MODE control statement and macro; the addition of control statement continuation, GETPARAM, and the GETMODE macro; and other minor changes to bring documentation into agreement with the released version 1.03 of the operating system.
F	December, 1978 - Revision F is the same as revision E with change packet E-01 added. No additional changes have been made.
F-01	January, 1979 - Change packet includes implementation of some features of BUILD; the addition of the BUFIN, BUFINP, BUFOUT, BUFOUTP, BUFEOF, and BUFEOD macros and other minor changes to bring documentation into agreement with the released version 1.04 of the operating system.
F-02	April, 1979 - Change packet includes the implementation of the DEBUG, RERUN, and NORERUN control statements, the RERUN, NORERUN, and BUFCHECK macros; changes to DUMP, DSDUMP, AUDIT, and ASSIGN control statements; implementation of job rerun and memory resident datasets. Other minor changes were made to bring documentation into agreement with the released version 1.05 of the operating system.
G	July, 1979 - Reprint with revision. This printing obsoletes all previous versions. Changes are marked with change bars. The changes bring this documentation into agreement with the released version 1.06 of the operating system.
G-01	December, 1979 - Change packet includes the implementation of the WAIT and NOWAIT options on the DISPOSE control statement; the addition of a new DUMP format and CFT Linkage Macros; and other minor changes to bring documentation into agreement with the released version 1.07 of the operating system.

Revision	Description
H	January, 1980 - Revision H is the same as revision G with change packet G-01 added. No additional changes have been made.
I	April, 1980 - Revision I is a complete reprint of this manual. All changes are marked by change bars. New features for version 1.08 of the operating system that are documented in this revision include: the addition of the CALL and RETURN control statements, job classes, the NA parameter on permanent dataset management control statements, the NRLS parameter on the DISPOSE control statement and PDD macro, and the CW parameter on the COMPARE control statement. Changes to the LDR control statement include the addition of the LLD, NA, USA, and I parameters and the new selective load directives. New documentation has been added for unblocked I/O, including descriptions of the READU and WRITEU macros. Other new macros include SETRPV, ENDRPV, DUMPJOB and the debugging aids SNAP, DUMP, INPUT, OUTPUT, FREAD, FWRITE, UFREAD, UFWRITE, SAVEREGS, and LOADREGS. Documentation on CRAY-1 interactive capabilities and changes to reflect the CRAY-1 S Series have also been added. Other changes were made to bring documentation into agreement with released Version 1.08 of the operating system. With this revision, the publication number has been changed from 2240011 to SR-0011.
I-01	October, 1980 - Change packet includes the implementation of the IOAREA, SETRPV, ROLL, and INSFUN macros and the IOAREA control statement; the addition of execute-only datasets including adding the EXO parameter to the SAVE and MODIFY control statements and the PDD macro; the lengthening of the TEXT parameter field; the addition of the DEB parameter to the LDR control statement; and a change to the formats of the UFREAD and UFWRITE macros. The DEBUG option allowing conditional execution of the SNAP, DUMP, INPUT, and OUTPUT macros has been implemented. Other minor changes were made to bring documentation into agreement with the released version 1.09 of the operating system.

Revision	Description
I-02	July, 1981 - This change packet includes changes to Job Control Language syntax; the addition of JCL block control statements for procedure definition (PROC, ENDPROC, &DATA, and prototype statement), conditional processing (IF, ELSE, ELSEIF, and ENDIF), and iterative processing (LOOP, EXITLOOP, and ENDLOOP); the addition of ROLLJOB, SET, LIBRARY, ECHO, PRINT, FLODUMP, and SYSREF control statements; the addition of CSECHO macro; the addition of CNS parameter to CALL statement, REPLACE parameter to BUILD statement, ARGSIZE parameter to ENTER macro, KEEP parameter to EXIT macro, USE parameter to ARGADD macro; the addition of the two JCL tables JBI and JST. Other minor changes were made to bring the documentation into agreement with the released version of 1.10 of the operating system.
J	February, 1982 - Reprint. This reprint incorporates revision I with change packets I-01 and I-02. No other changes have been made.
J-01	June, 1982 - This change packet includes the following additions: magnetic tape characteristics, temporary and local dataset clarification, mass storage permanent datasets, magnetic tape permanent datasets, tape I/O formats, interchange format, transparent format, new accounting information, **gn=nr* parameter, several CHARGES parameters, the OPTION control statement, procedure definition, HOLD parameter, new information to the ACCESS control statement, new tape dataset parameters, tape dataset conversion parameters, SUBMIT job control statement, PDSDUMP and PDSLOAD sample listings, SID parameter on the LDR control statement, new loader errors, relocatable overlays, CONTRPV macro, SUBMIT macro, unrecovered data error information, POSITION macro, new PDD macro parameters, the LDT macro, and new glossary terms. The information formerly in Appendix C is now in the COS EXEC/STP/CSP Internal Reference Manual, publication SM-0040. Other miscellaneous technical and editorial changes were made to bring the documentation into agreement with version 1.11 of the operating system.
K	July, 1982 - Reprint. This reprint incorporates revision J with change packet J-01. No other changes have been made.

Revision	Description
L	July, 1983 - Revision L is a rewrite of this manual. Extensive editorial changes have been made, including moving macro information which was in part 3 to publication SR-0012, Macro and Opdefs Reference Manual. Other major reorganization has occurred. Part 3 now contains job control language structures. Information has been added on interactive job processing and job step abort processing. Major new features documented include enhanced support of tape datasets, the FETCH control statement, memory management, enhancements to COS security, permanent dataset privacy, and support of the CRAY X-MP Computer System. Miscellaneous editorial and technical changes have been made to bring the documentation into agreement with version 1.12 of the operating system. All previous versions are obsolete.
L-01	October, 1983 - This change packet describes two new ACCOUNT control statement parameters: APW and NAPW. The use of APW and NAPW, and their interrelationship with existing parameters on ACCOUNT, are also explained. A new parameter on the AUDIT control statement, ACC, is described. In addition, illustrative information is provided on how the OWN parameter of the AUDIT utility affects output listings.
L-02	February, 1984 - This change packet supports the COS 1.13 release. It includes editorial and technical amendments to information which had been included in previous versions of this manual. The contents reflect new multitasking capabilities. Additional information has been included for coding the CALL statement. New parameters have also been documented in this manual for foreign dataset processing, particularly on the ASSIGN and ACCESS control statements. The LDR statement has been modified considerably; RELEASE, SAVE, MODIFY, DELETE, PERMIT, ACQUIRE and PDSLOAD also have new parameters. Furthermore, new information is included for managed memory capabilities, the EXITIF control statement block identifier, the COPYU utility for unblocked datasets, and new error codes for reprieve processing.

Revision	Description
M	December, 1984 - This reprint with revision describes many technical changes to COS for the 1.14 release, including contiguous disk allocation and the tape features multitape mark, online tape ring processing, partial IBM multifile, special end-of-volume processing, and superblock size. The revision describes software to support four-processor CRAY X-MPs and systems with up to 8 million words of memory. Appendix B provides instructions for Subsystem Support: interjob communication, user channel access, and event recall. This revision also documents the Integrated Support Processor (ISP). Note that ISP code will be released later. This revision contains several format changes. To increase the accuracy of the tables and related information in appendix A, the section is printed as generated by the system. In the body of the manual the "parts" have been removed and the sections numbered consecutively. Material in the four sections of part 3 has been consolidated into one section, 16. All previous printings are obsolete.

PREFACE

This manual describes the external features of the Cray Operating System (COS). It is intended as a reference document for all users of COS. The manual deals with three aspects of COS:

- Job processing. Sections 1 through 5 discuss the fundamentals of creating and running jobs on a Cray Computer System. These sections describe the system components, storage of information on a Cray Computer, and job processing. They also introduce COS job control and describe the use of libraries.

- Job control statements. Sections 6 through 15 describe each COS job control statement and give the format of each with an explanation of its function.

- Control statement structures. Section 16 describes the control statement block structures available with COS. Examples at the end of the section demonstrate the COS control statement procedure substitution process.

Other CRI publications that may be of interest to the reader are:

<u>Products and Utilities</u>

SR-0010	Software Tools Reference Manual
SR-0013	UPDATE Reference Manual
SG-0055	Text Editor (TEDI) User's Guide
SG-0056	Symbolic Interactive Debugger (SID) User's Guide
SR-0066	Segment Loader (SEGLDR) Reference Manual
SR-0073	Cray Simulator (CSIM) Reference Manual
SR-0074	SORT Reference Manual

<u>Languages</u>

SR-0000	CAL Assembler Version 1 Reference Manual
SR-0009	FORTRAN (CFT) Reference Manual
SR-0012	Macros and Opdefs Reference Manual
SR-0014	Library Reference Manual
SR-0060	Pascal Reference Manual

Hardware

HR-0004	CRAY-1 Hardware Reference Manual
HR-0029	CRAY-1 S Series Mainframe Reference Manual
HR-0030	CRAY I/O Subsystem Reference Manual
HR-0032	CRAY X-MP Series Mainframe Reference Manual
HR-0064	CRAY-1 M Series Mainframe Reference Manual
HR-0630	Mass Storage Subsystem Hardware Reference Manual

Miscellaneous

SR-0039	CRAY-OS Message Manual
SN-0222	Multitasking User Guide

CONTENTS

APPENDIX SECTION

FIGURES

FIGURES (continued)

TABLES

TABLES (continued)

INTRODUCTION TO JOB PROCESSING 1

The Cray Operating System (COS) is a multiprogramming, multiprocessing, and multitasking operating system for Cray Computer Systems. The *operating system* provides for efficient use of system resources by monitoring and controlling the flow of work presented to the system in the form of jobs. The operating system optimizes resource usage and resolves conflicts when more than one job is in need of resources.

COS is a collection of programs residing in Cray mainframe Central Memory or on system mass storage following *startup* of the system. (Startup is the process of bringing the Cray Computer System and the operating system to an operational state.)

Jobs are presented to the Cray Computer System by one or more computers referred to as *front-end computers* (also referred to as *stations* in Cray Research manuals). A front-end computer can be any of a variety of computer systems. Software executing on the front-end computer system is beyond the scope of this publication.

COS includes linkages providing for the initiation and control of interactive jobs and data transfers between the Cray Computer System and front-end terminals. These features are available only where supported by the front-end system.

The FORTRAN compiler (CFT), library routines, the CAL assembler, and the UPDATE source maintenance program are described in separate publications.

HARDWARE REQUIREMENTS

The Cray Operating System (COS) executes on the basic configuration of any CRAY-1 or CRAY X-MP Computer System. Each computer system contains the following components:

- One or more Central Processing Units (CPUs)
- Central Memory
- An I/O Subsystem (IOS) or a minicomputer-based Maintenance Control Unit (MCU). The I/O Subsystem performs all required Maintenance Control Unit functions.

- A Mass Storage Subsystem. The Mass Storage Subsystem consists of disk drives, an optional Solid-state Storage Device (SSD), and IOS Buffer Memory (BMR).

- An optional IBM-compatible tape subsystem. The tape subsystem requires that an I/O Subsystem be present.

The I/O Subsystem consists of from two to four I/O processors and one-half million, one million, four million, or eight million words of shared Buffer Memory. The optional tape subsystem is composed of at least one block multiplexer channel, one tape controller, and two tape units. The tape units supported are IBM-compatible 9-track, 200 ips, 1600/6250 bpi devices.

Figure 1-1 illustrates a basic system configuration. For more information about CRAY-1 or CRAY X-MP hardware characteristics, refer to the appropriate mainframe reference manual listed in the preface.

SYSTEM INITIALIZATION

COS is loaded into Central Memory and activated through a system startup procedure performed at the I/O Subsystem or MCU. At startup, linkage to the Permanent Dataset Catalog (DSC) is reestablished on mass storage. All permanent mass storage datasets are recorded in the DSC; thus, permanent datasets survive startup and the user can always assume that they are present. See section 2 of this manual for more information on datasets.

CENTRAL MEMORY ASSIGNMENT AND CHARACTERISTICS

Central Memory is shared by COS, jobs running on the Cray mainframe, dataset I/O buffers, and system tables associated with those jobs. COS allocates resources to each job, when needed, as these resources become available. As a job progresses, information is transferred between Central Memory and mass storage. These transfers can be initiated by either the job or by COS.

Figure 1-2 illustrates the assignment of memory to COS and to jobs.

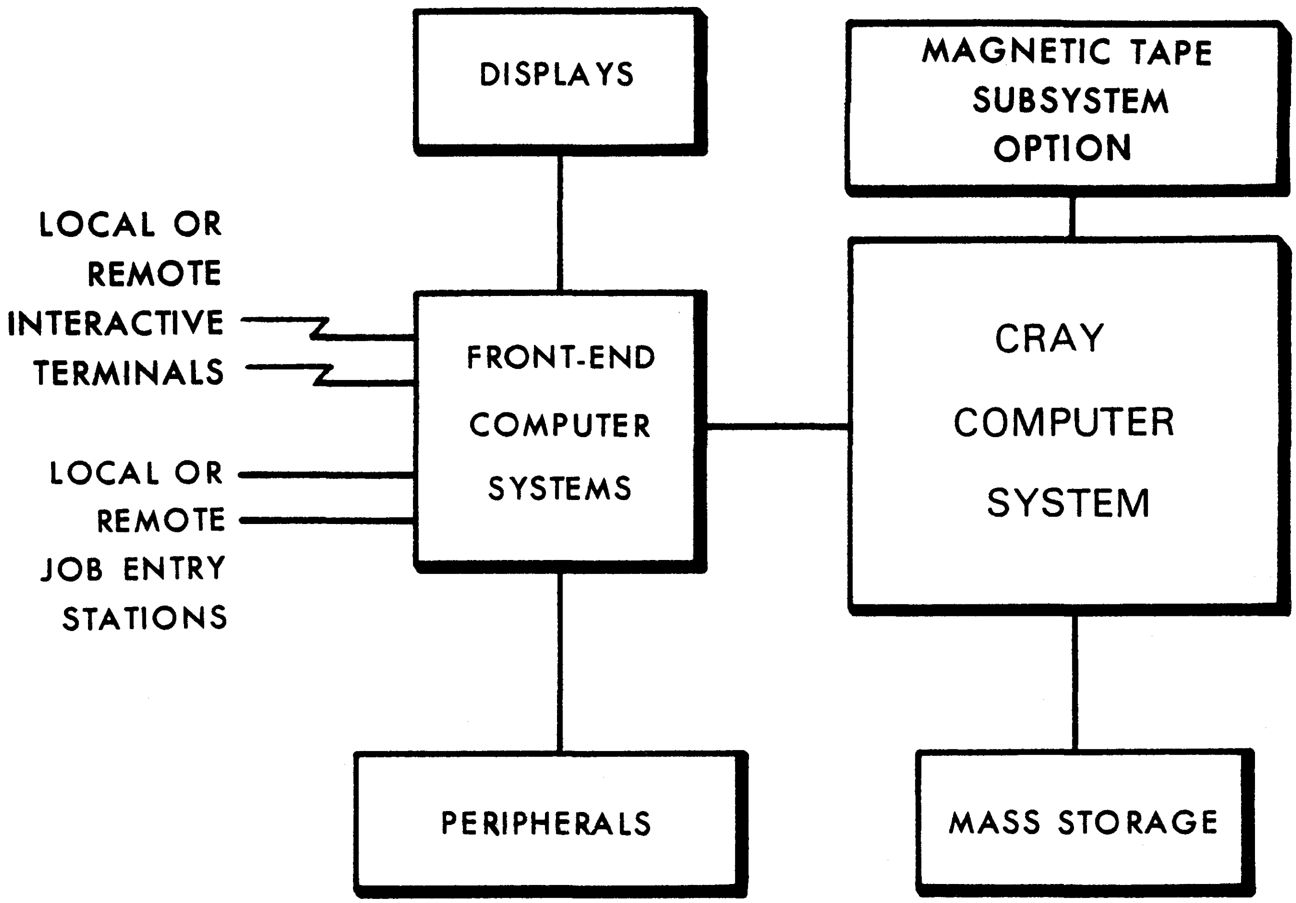

Figure 1-1. Cray Computer System configuration

MEMORY-RESIDENT COS

COS occupies two areas of Central Memory. The memory-resident portion of the operating system occupying lower memory consists of Exchange Packages, the System Executive (EXEC), the System Task Processor (STP), and optionally the Control Statement Processor (CSP). The memory-resident portion of the operating system occupying extreme upper memory contains station I/O buffers, space for the system log buffer, and Permanent Dataset Catalog (DSC) information and buffers.

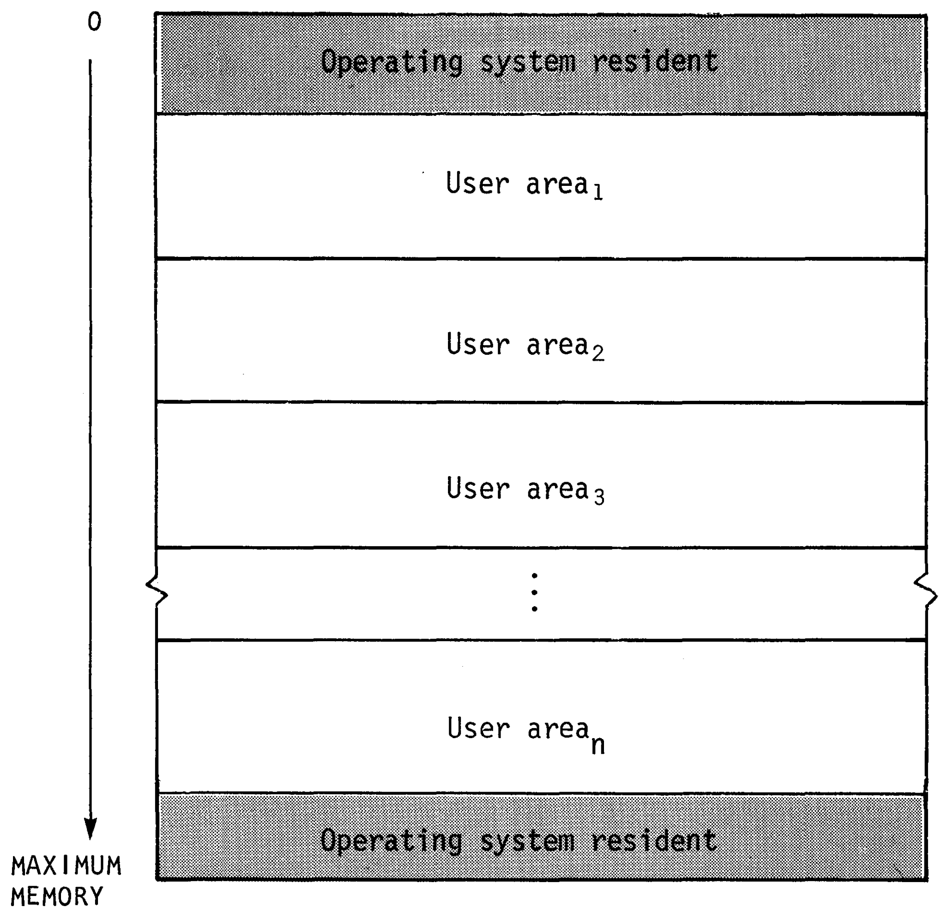

Figure 1-2. Central Memory assignment

USER AREA OF MEMORY

COS assigns every job a *user area* in Central Memory. The user area consists of a Job Table Area (JTA) and a user field.

Job Table Area - JTA

For each job, the operating system maintains an area in memory that contains the parameters and information required for monitoring and managing the job. This area is called the Job Table Area (JTA). Each active job has a separate Job Table Area adjacent to the job's user field. The Job Table Area is not accessible to the user, although it can be dumped for analysis (see section 13 of this manual).

User field

The *user field* for a job is a block of memory immediately following the job's JTA. The user field is always a multiple of 512 words. The beginning or *Base Address* (BA) and the end or *Limit Address* (LA) are set by the operating system. The maximum user field size is specified by a parameter on one of the job control statements (see section 6) or by installation-defined default. A user can request changes in user field size during the course of a job.

Compilers, assemblers, system utility programs, and user programs are loaded from mass storage into the user field and are executed in response to control statements in the job deck. Each load and execution of a program is referred to as a *job step*.

A detailed description of the contents of the user field is given in section 3 of this manual. Briefly, however, the first 200_8 words of the user field are reserved for an operating system/job communication area known as the Job Communication Block (JCB). Programs are loaded starting at BA+200_8 and reside in the lower portion of the user field. The upper portion of the user field contains tables and dataset I/O buffers. The user field addressing limit is equal to LA-1.

Memory addresses for instructions and operands are relative to BA. The Cray mainframe adds the contents of BA to the address specified by a memory reference instruction to form an absolute address. A user cannot reference memory outside of the user field as defined by the BA and LA register contents; LA-1 is the user limit. (Refer to the appropriate mainframe hardware reference manual noted in the preface for more information.)

MASS STORAGE CHARACTERISTICS

All information maintained on mass storage by the Cray Operating System (except specific pre-allocated areas such as the Device Label) is organized into quantities of information known as *datasets*. In general, the user need not be concerned with the physical transfer of data between the disks and memory nor with the exact location and physical form in which datasets are maintained on mass storage. COS translates the user's logical requests for data input and output into disk controller functions automatically.

Each disk storage unit contains a device label, datasets, and unused space to be allocated to datasets. The *device label* notes usable and unusable (unflawed and flawed) space on the disk unit and designates one of the devices as the Master Device. The *Master Device* is the disk storage unit containing a table known as the *Dataset Catalog* (DSC), which contains information for maintaining permanent datasets.

To the user, mass storage *permanent datasets* are always present and available on mass storage. This permanence is achieved through techniques permitting the datasets noted in the DSC to be recovered or reestablished in the event of system failures. Portions of COS, such as the loader, utility programs, the compiler, the assembler, and library maintenance and generation routines, reside in permanent datasets accessible by user jobs at any time.

Datasets containing job input decks and output from jobs also reside on mass storage. Because these datasets are listed in the Dataset Catalog they are also regarded as permanent. This designation is somewhat misleading since their permanence is by definition rather than by tenure in the system. That is, the input dataset is permanent from the time it is staged from the front-end system to the Cray Computer System until the job terminates. Output datasets being disposed to a front end are permanent from job termination (or whenever the disposition was initiated) until the disposition is complete. The permanence of these system-defined datasets allows them to be recovered along with other permanent datasets after a system failure.

Any user job can create a mass storage permanent dataset. It can be subsequently accessed, modified, or deleted by any other job having correct access privileges and producing the correct permission control words when attempting to associate it with the job. Permission control words are defined at the time the dataset is designated as permanent (that is, *saved*).

A permanent dataset ceases to be permanent when a user with the correct permission control word deletes it. This deletion notifies COS that the space occupied by the dataset is no longer permanent. However, the space is still reserved by the dataset until it is released by the user (see sections 8 and 10, respectively, for information on the RELEASE and DISPOSE control statements).

In addition to the various permanent datasets, mass storage is used for temporary datasets. A *temporary dataset* is created by the job using it and remains temporary unless it is designated as permanent, released, or disposed to a front end by the job. A temporary dataset neither saved as permanent nor disposed of is termed a *scratch dataset* and ceases to exist when the job releases it or terminates.

COS allocates space to datasets as needed by tracks. Storage assigned to a single dataset can be noncontiguous and can even be on multiple disk units. Default and maximum sizes for datasets are defined by system parameters. The user has limited control over the allocation of storage to a dataset through the ASSIGN control statement.

MAGNETIC TAPE CHARACTERISTICS

An I/O Subsystem can include an Auxiliary I/O Processor (XIOP) with the capability of addressing up to 16 block multiplexer channels of tape units. Each block multiplexer channel can be attached to IBM-compatible control units and tape units in a variety of configurations. The block multiplexer channels communicate with the control units and tape units to allow reading and writing data that can also be read and written on IBM-compatible CPUs. The physical characteristics of tape devices are summarized in table 1-1. The block sizes in this table are used by the COS tape system for transparent-format tape datasets (described in section 2).

Table 1-1. Physical characteristics of 200 ips, 9-track tape devices

Density (bits/inch)	Transfer rate (kilobytes/sec)	Data/2400 ft. reel (megabytes)	Percent of reel containing data	Block size (bytes)
6250	1170	168	94	32768
1600	300	43	94	16384

DATASETS 2

Nearly all information maintained by the Cray Operating System (COS) is organized into quantities of information known as *datasets*. The following are some of the more important factors to remember about datasets.

- The dataset *medium* is the type of physical device on which the dataset resides.
- The dataset *structure* is the logical organization of the dataset.
- The dataset *longevity* is the retention period for the dataset.
- A dataset must be *local* to be usable.
- The dataset *disposition code* tells the operating system what action to take when the dataset is no longer local.
- Each dataset is known by its *dataset name*.
- Datasets are read and written using operating system requests (user I/O interfaces).

DATASET MEDIUM

Datasets can be classified by medium, as follows:

- Mass storage datasets
- Memory-resident datasets
- Interactive datasets
- Magnetic tape datasets

MASS STORAGE DATASETS

All datasets, unless otherwise specified, reside on Cray mass storage, that is, on mass storage attached directly to the mainframe or to the I/O Subsystem.

MEMORY-RESIDENT DATASETS

Some datasets can be specified by the user as memory-resident datasets. A *memory-resident dataset* is wholly contained within one buffer (see BS parameter on the ASSIGN control statement in section 8 of this manual) and remains in memory at all times. Such a dataset ordinarily occupies no mass storage. A memory-resident dataset is normally a temporary dataset; however, a mass storage permanent dataset can be declared memory resident.

A dataset can be declared memory resident to reduce the number of I/O requests and disk blocks transferred. Memory residence is particularly useful for intermediate datasets not intended to be saved or disposed to another mainframe. All I/O performed on a memory-resident dataset takes place in the dataset buffers in memory and the contents of the buffers are not ordinarily written to mass storage. Such a dataset cannot be made permanent, nor may it be disposed to another mainframe, unless copied to mass storage.

Normally, a memory-resident dataset is empty until written on. If an existing dataset is declared memory resident, it is loaded when the first read occurs. A user attempting to write to a memory-resident dataset must have write permission. However, as long as the buffer does not appear full, no actual write to mass storage ever occurs. Therefore, changes made to an existing dataset declared memory resident are not reflected on the mass storage copy of the dataset.

A memory-resident dataset must be defined through an ASSIGN control statement containing the MR parameter or through an F$DNT call to the system. If the F$DNT call is used, the Dataset Definition List (DDL) supplied should specify DDMR=1. (See the description of the ASSIGN control statement in section 8 of this manual.) In addition, the buffer size parameter should specify a buffer large enough to contain the entire dataset plus one block.

If at any time the system I/O routines are called to write to the dataset and the buffer appears to be full, the dataset ceases to be treated as memory resident, the buffer is flushed to mass storage, and all memory-resident indicators for the dataset are cleared.

Magnetic tape, execute-only, and interactive datasets cannot be declared memory resident.

INTERACTIVE DATASETS

A dataset can be specified as interactive by an interactive job, provided that interactive datasets are supported by the front end. Batch users cannot create interactive datasets. An interactive dataset differs from a local dataset in that a disk image of the dataset is not maintained.

Instead, records are transmitted to and from a terminal attached to a front-end station. Record positioning (for example, REWIND or BACKSPACE) is not possible.

Interactive datasets can be created by interactive jobs through the use of the ASSIGN control statement or F$DNT system call.

MAGNETIC TAPE DATASETS

A *magnetic tape dataset* is available to any job declaring tape resource requirements on the JOB statement and specifying the appropriate information on its ACCESS request.

To gain access to an existing tape dataset for reading and/or rewriting, the correct file identifier (permanent dataset name), the desired device type, and, optionally, a volume identifier list must be specified. The volume identifier list can consist of 1 to 255 volume identifiers. If the permanent dataset name (PDN) is omitted from the ACCESS request, the local dataset name is used as the file identifier.

To gain access to a tape dataset for creating, the file identifier, desired device type, and the NEW parameter option must be specified on the ACCESS request. If no file identifier is present, the local dataset name is used. If the volume identifier list is missing from the access request, it is called a *non-specific volume allocation*. A *specific volume allocation* occurs when the volume identifier list is present at the time of the access request. New tape datasets must be written to before a read is allowed.

Other options describing the tape dataset are available from the access request. See the ACCESS control statement description (section 9 of this manual) for more details. Using other parameter options allows more efficient tape dataset descriptions.

COS automatically switches volumes during dataset processing unless user EOV processing is requested, and returns to the first volume of a multivolume dataset in response to a REWIND command. If a permanent write error occurs when trying to write a tape block for the user, COS automatically attempts to close the current volume and continues to the next volume.

The COS tape system uses Buffer Memory as a tape block buffering area so that the job's I/O buffer need not be as large as the tape block (as with other operating systems). This technique can result in significant memory savings whenever large tape blocks are being processed and in increased transfer rates whenever smaller blocks are being processed. The advantage in having a large COS buffer is a reduction in the overhead in the tape subsystem.

USER TAPE END-OF-VOLUME PROCESSING

The user tape end-of-volume (EOV) feature allows the user to gain control at tape end of volume and perform special EOV and BOV processing. The macros used are SETSP, STARTSP, ENDSP, TAPESTAT and CLOSEV. These macros are used on individual datasets. If EOV processing is needed for more than one dataset, the macros must be issued for each tape dataset. Refer to CRI publication SR-0012, Macros and Opdefs Reference Manual for more information.

The user instructs the system to perform EOV processing by issuing the SETSP macro (with the ON option) after a tape dataset is opened. Using SETSP with the OFF option informs the system that EOV processing is no longer needed. The CLOSE macro also terminates EOV special processing.

To test that the tape dataset is at EOV, you must use the TAPESTAT macro after every READ, WRITE, and SYNCH macro. Not all macros that result in I/O operations return EOV status; for example, the CLOSE, POSITION, and REWIND macros do not return EOV status. For output datasets, the user should use the SYNCH macro to flush the buffers and check to see if EOV has been encountered before using such macros.

After EOV is encountered, the user starts EOV processing by issuing the STARTSP macro. During EOV processing the user may do read, write, and position operations. Volume switching is done by issuing the CLOSEV macro. When EOV processing is completed, the ENDSP macro notifies the system to return to normal processing.

During EOV processing, no read ahead is performed. Data blocks are read one at a time. Also, a position request with relative block number is positioned from the current tape position, not from the last I/O block.

For an output dataset, the data in the IOP buffer when EOV is encountered is considered part of the dataset and may be read during EOV processing. Once the data is read, it is no longer part of output data. Because no read ahead is performed during EOV processing, the program may position backwards and read only the blocks on the tape. If this is the case, the data in the IOP buffer is written to tape when the ENDSP macro is issued.

The use of the CLOSEV macro is not restricted to the EOV routine. The CLOSEV macro may be issued by the user anytime during dataset processing. This macro allows the user to terminate an output tape anywhere and continue the dataset on the next tape. It also allows the user to read part of a tape and switch to the following tape.

TAPE MARK PROCESSING BY TQM

Three label types are available that allow tape marks to be embedded in data. They also allow a user to process the tape mark as data and continue processing after its occurrence is detected. The label types are specified on the ACCESS statement and allow processing of field ANSI labels (FAL), field standard IBM labels (FSL), and field nonlabeled (FNL) tapes.

When TQM recognizes a tape mark in the data, it translates it to an EOF record control word and puts it in the data. The user gets a tape mark indication when processing the data using the TAPESTAT macro and then is able to continue normal processing. The recognition of end-of-file conditions is the responsibility of the user.

To keep the program from running off the end of the tape on reads, the software stops processing (that is, no reading ahead occurs) when a tape mark is detected and does not move forward until the user catches up to the tape position, recognizes the tape mark, and issues further read commands.

Any attempt to position past a tape mark (using the POSITION macro) results in the tape moving until the tape mark is encountered. At that point, tape movement stops and the user job gets control. A residual record count is returned to find the position on tape. The tape is physically positioned after the tape mark just encountered.

For input, all field format tapes (FAL/FNL/FSL) are processed for labels in the same way. At BOT, if a label is encountered it is validated based on its type. If no label is found, there is no validation. When a tapemark is detected, the system checks the next record for a EOV1 or EOF1 trailer label. If EOV1 is found, the system performs an automatic volume switch. If an EOF1 is found, the system performs end-of-data processing. If neither EOV1 or EOF1 is encountered, the tape is left positioned immediately following the tape mark ready for the next read. Tapes not terminated with either SL or AL standard labels must be terminated by the program using CLOSE or CLOSEV system calls.

For output, field format tapes are labeled based on the LB parameter on the ACCESS statement. End-of-volume labels are processed when either the EOT reflective marker is sensed or when the user program calls the CLOSEV routine. End-of-file labels are written when the dataset is closed, rewound, or released.

DATASET STRUCTURE

COS supports several dataset structures:

- Blocked format
- Interactive format
- Unblocked format
- Tape formats (interchange or transparent)

BLOCKED FORMAT

Blocked format is used by default for external types of datasets, such as user input and output datasets. Record positioning requires a blocked format. The blocked format adds control words to the data to allow for processing of variable-length records and to allow for delimiting of levels of data within a dataset. A blocked dataset can be composed of one or more files, which are, in turn, composed of one or more records. Figure 2-1 illustrates the data hierarchy within a dataset.

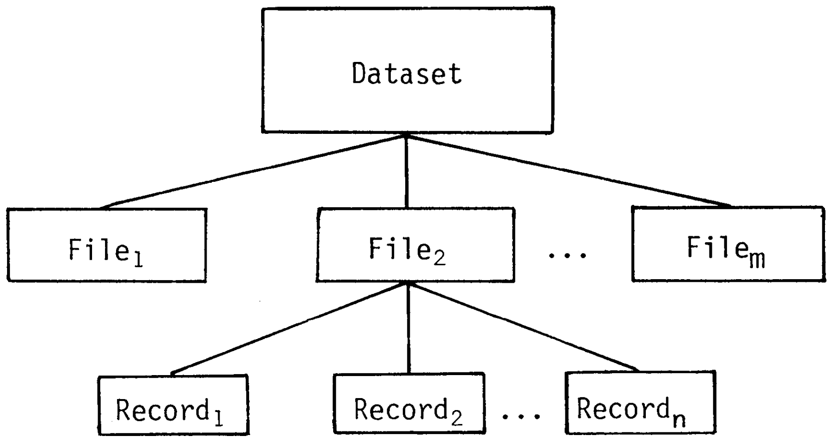

Figure 2-1. Data hierarchy within a dataset

The data in a blocked dataset can be coded and/or binary. Blanks are normally compressed in blocked coded datasets. Each block consists of 512 words. Blocked datasets use two types of control words: block and record.

Blank compression

Blank fields can be compressed for blocked coded files. Blank field compression is indicated by a blank field initiator code followed by a count. The default blank field initiator code is defined by the installation parameter I@BFI which is either an ASCII code or 777_8 indicating that blank compression will not be done. Blank compression can be inhibited using an ASSIGN statement parameter or an F$DNT system call. A blank field of 3 through 96 characters is compressed to a 2-character field. The count is biased by 36_8; the actual character count is limited to $41_8 \leq$ *character count* $\leq 176_8$ (the ASCII graphics).

Block control word

The block control word (BCW) is the first word of every 512-word block. The format of a block control word is depicted in figure 2-2.

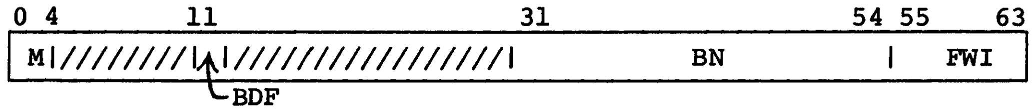

Figure 2-2. Format of a block control word

Field	Bits	Description
M	0-3	Type of control word (for block control word, M=0)
BDF	11	Bad Data flag; indicates the following data, up to the next control word, is bad. This flag is set by the I/O Subsystem for magnetic tape datasets in interchange format.
BN	31-54	Block number. Designates the number of the current data block. The first block in a dataset is block 0.
FWI	55-63	Forward index. Designates the number of words (starting with 0) to the next record control word or block control word.

Record control word

A record control word (RCW) occurs at the end of each record, file, or dataset. The format of a record control word is illustrated in figure 2-3.

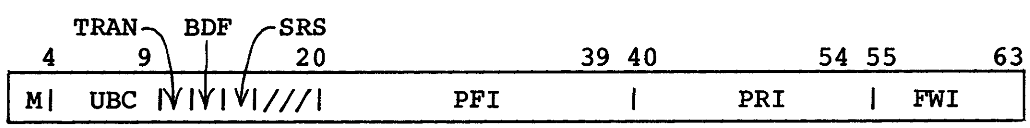

Figure 2-3. Format of a record control word

Field	Bits	Description
M	0-3	Type of control word: 10_8 End-of-record (EOR) 16_8 End-of-file (EOF) 17_8 End-of-data (EOD)

Field	Bits	Description
UBC	4-9	Unused bit count. For end-of-record, UBC designates the number of unused low-order bits in the last data word of the record terminated by the end-of-record. For end-of-file and end-of-data RCWs, this field is 0. The data area protected by UBC must be zero-filled.
TRAN	10	Transparent record field; used for an interactive output dataset only. If set, substitution of end-of-record RCWs is suppressed.
BDF	11	Bad Data flag; indicates the following data, up to the next control word, is bad. This flag is set by the I/O Subsystem for magnetic tape datasets in interchange format. If flag is set, an irrecoverable error was encountered in following data.
SRS	12	Skip remainder of sector; indicates that the next control word to follow is a BCW and the data after this RCW is not to be processed. This is used only in tape dataset processing.
PFI	20-39	Previous file index. This field contains an index modulo 2^{20} ($20{,}000{,}000_8$) to the beginning of the file. The index is relative to the current block such that if the beginning of the file is in the same block as this RCW, the PFI is 0.
PRI	40-54	Previous record (RCW) index. This field contains an index modulo 2^{15} ($100{,}000_8$) to the block where the current record starts. The index is relative to the current block such that if the first word of data in this record is in the same block as this RCW, PRI is 0.
FWI	55-63	Forward word index. This field points to the next control word (RCW or BCW) and consists of a count of the number of data words up to the control word (that is, if the next word is an RCW or BCW, FWI is 0).

Disregarding block control words occurring at 512-word intervals in a dataset, RCWs have the following logical relationship in a dataset.

An end-of-record RCW immediately follows the data for the record it terminates. If the record is null, that is, if it contains no data, an end-of-record RCW can immediately follow an end-of-record or end-of-file RCW or can be the first word of the dataset.

An end-of-file RCW immediately follows the end-of-record RCW for the final record in a file. If the file is null, that is, if it contains no records, the end-of-file RCW can immediately follow an end-of-file RCW or can be the first word of the dataset.

An end-of-data RCW immediately follows the end-of-file RCW for the final file in the dataset. If the dataset is null, the end-of-data RCW can be the first word on the dataset.

The typical dataset has many end-of-record RCWs per block. An example of dataset control words is illustrated in figure 2-4. In this example, a dataset is contained within four physical sectors, each beginning with a BCW (thus the four BCWs in this example are numbered 0, 1, 2, 3). The dataset contains four files shown as F1, F2, F3, and F4. F1 contains the four records shown as R1 through R4; F2 contains records R5 through R7; F3 contains no records at all; F4 contains record R8.

INTERACTIVE FORMAT

Interactive format closely resembles blocked format; however, each buffer begins with a block 0 BCW. Each record transmitted to or from COS by an F$RDC or an F$WDC call must contain a single record consisting of a BCW, data, and an end-of-record RCW.

Two formats for interactive output can be assigned when the dataset is created: character blocked and transparent. Character blocked mode is the default. In character blocked mode, an end-of-record RCW is interpreted as a line feed or a carriage return. In transparent mode, the end-of-record RCW is ignored and the user is responsible for supplying carriage control characters.

UNBLOCKED FORMAT

Dataset I/O can also be performed using unblocked datasets. The data stream for unblocked datasets does not contain Cray Operating System record control words (RCWs) or block control words (BCWs).

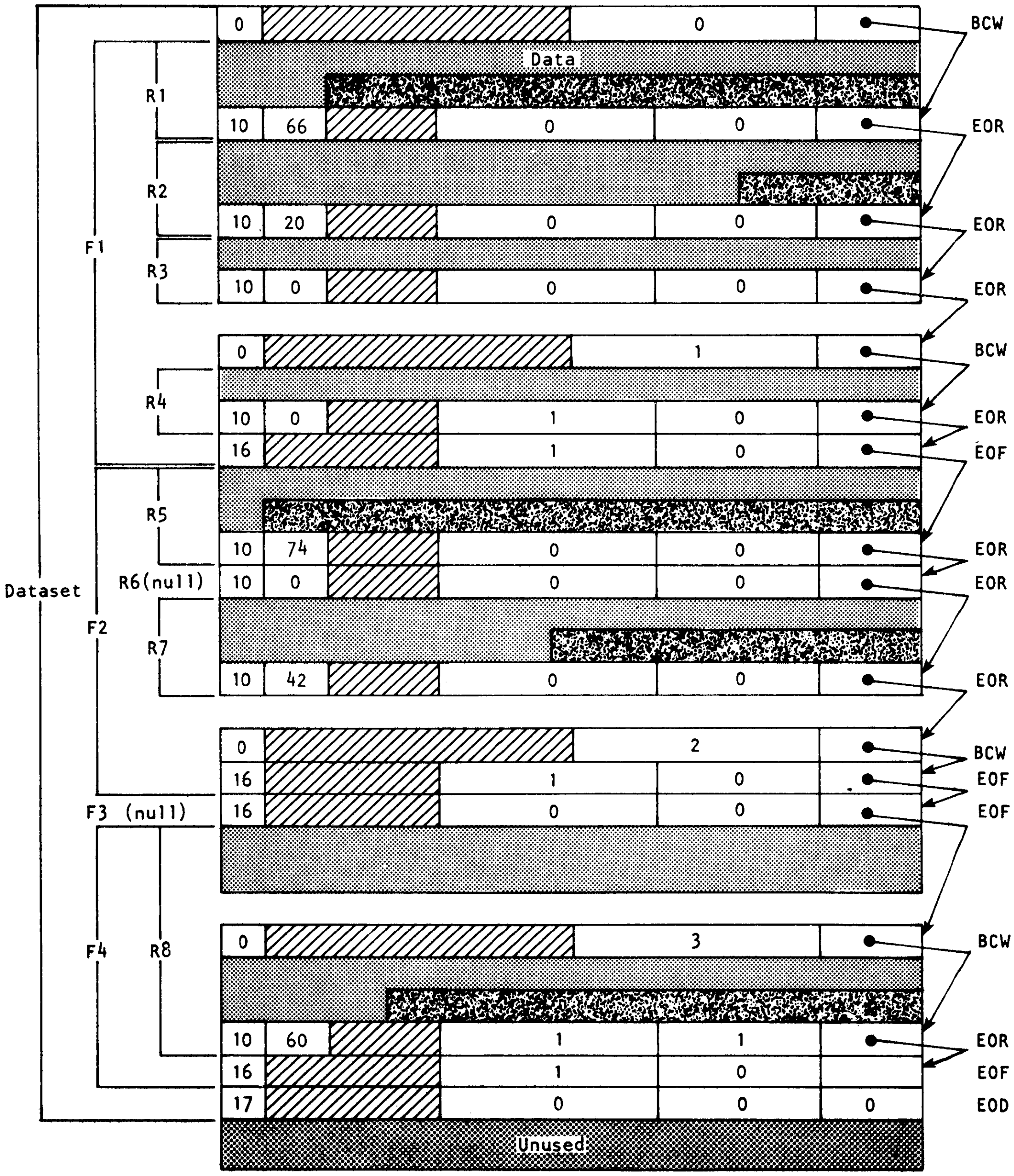

Figure 2-4. Example of dataset control words
(octal values shown)

The system does not allocate buffers in the job's I/O buffer area for unblocked datasets; the user must specify an area for data transfer. When a read or write is performed on an unblocked dataset, the data goes directly to or from the user data area without passing through an I/O buffer. The word count of data to be transferred must be a multiple of 512.

Unblocked I/O cannot be performed on an interchange format tape dataset.

TAPE FORMATS

Tape datasets are written and read on tape volumes. A *tape volume* is a reel of tape. A tape volume is also known as a dataset section (for example, in FSEC= on the ACCESS statement).

Data is read or written in tape blocks. A *tape block* is a unit of data recorded on magnetic tape between two consecutive interblock gaps. The size of tape blocks can vary from one byte to an installation-defined maximum.

Tape datasets can be read or written using two different formats: *interchange* or *transparent*. Tape datasets can also be labeled or unlabeled.

Interchange format

Interchange format facilitates reading and writing tapes that are also to be read or written on other vendors' systems. In *interchange format*, each tape block of data corresponds to a single logical record in COS blocked format (that is, the data between record control words).

In interchange format, tape block lengths can vary up to an installation-defined maximum which cannot exceed 1,048,576 bytes (131,072 64-bit words). It is recommended that the maximum block size not exceed 100 to 200 kilobytes. Blocks exceeding these sizes may require special operational procedures (such as the use of specially prepared tape volumes having an extended length of tape following the end-of-tape (EOT) reflective marker) and yield little increase in transfer rates or storage capacity.

When a tape dataset is read in interchange mode, physical tape blocks are represented in the user's I/O buffer with block control words (BCWs) and record control words (RCWs) added by COS. The data in each tape block is terminated by an RCW. The unused bit count field in the RCW indicates the amount of data in the last word of the tape block that is not valid data. A BCW is inserted before every 511 words of data, including the RCWs. The formats of RCWs and BCWs are described previously in this section and shown in figures 2-2 and 2-3.

Figure 2-5 depicts a tape dataset in interchange format. Tape blocks within tape label groups are not included in this format. The end of the dataset is represented by an end-of-file (EOF) RCW followed by an end-of-data (EOD) RCW.

When a tape dataset is written in interchange format, the data must be in the I/O buffer in the user field in COS blocked format. The data in each logical record is written as a single tape block. BCWs and RCWs are not recorded on tape. BCWs within a record are discarded and the unused bits and terminating RCW are also discarded. The unused bit count must be a multiple of 8. Tape datasets written in interchange mode must consist of a single file (single EOF RCW). Multiple-file tape datasets are not supported in interchange mode.

Transparent format

In *transparent format* (disk image), each tape block is a fixed multiple of 4096 bytes (512 words), generally based on the dataset density (that is, 16,384 bytes at 1600 bpi and 32,768 bytes at 6250 bpi). The data in the tape block is transferred unaltered between the tape and the I/O buffer in the user field; no control words are added on reading or discarded on writing. In transparent mode, the data can be in COS blocked format or COS unblocked format. Transparent format tapes are not generally read or written by other vendors' equipment.

DATASET LONGEVITY

Permanent datasets are retained by the operating system until instructed otherwise. All other datasets are considered temporary.

TEMPORARY DATASETS

A *temporary dataset* is available only to the job that created it. Temporary datasets can be created in two ways: either explicitly by use of the ASSIGN control statement, or implicitly upon first reference to a dataset by name or unit number on an I/O request or an OPEN macro call.

A temporary mass storage dataset is empty until written on. Rewind or backspace of the dataset is necessary before it can be read. A temporary dataset can be made permanent by use of the SAVE control statement. If the dataset is not made permanent, it is released at job termination or by the specific RELEASE function request and its mass storage made available to the system.

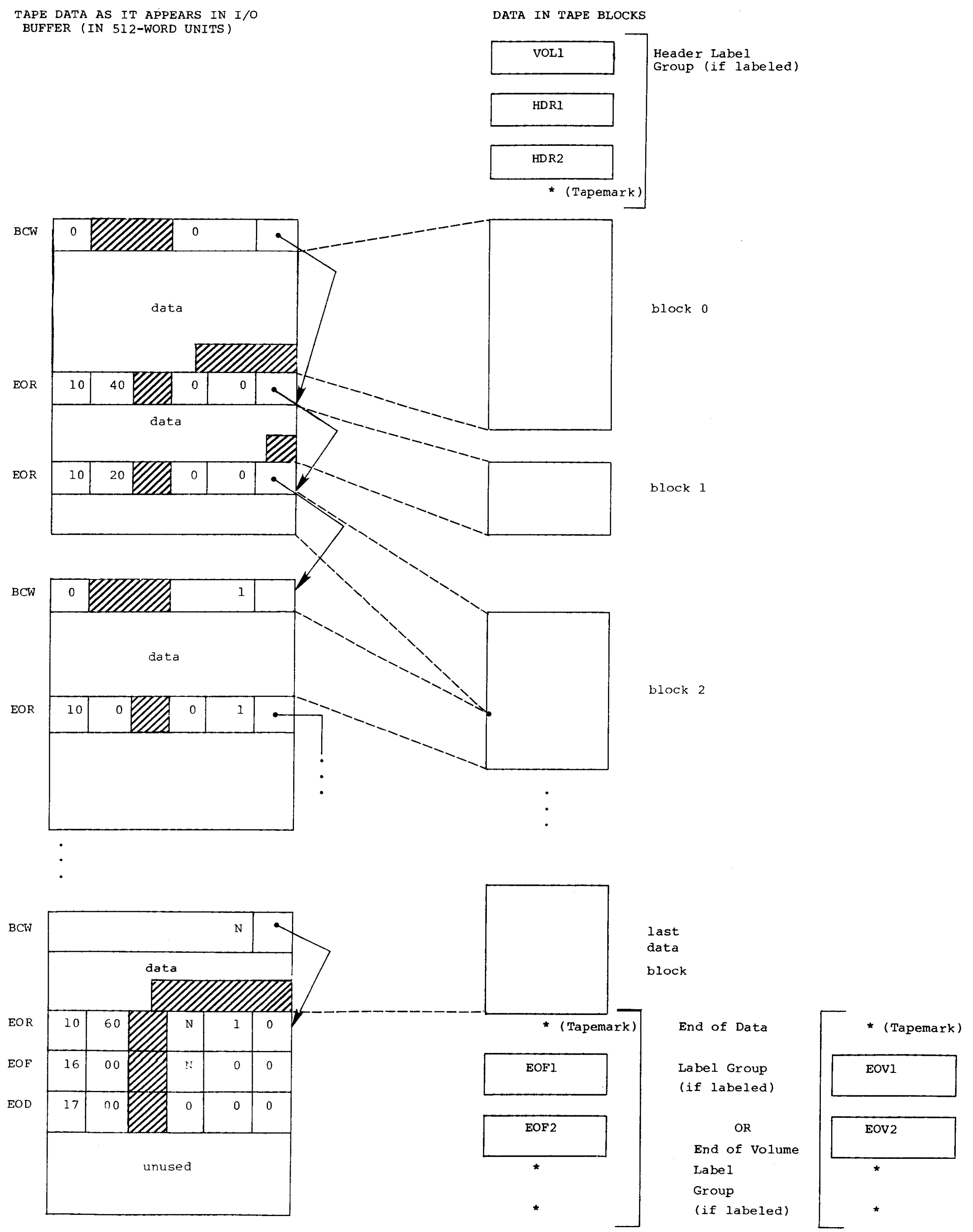

Figure 2-5. Interchange-format tape dataset (octal values shown)

PERMANENT DATASETS

Only mass storage or magnetic tape datasets can be permanent.

Magnetic tape permanent datasets

Tape datasets are discussed under Dataset Media earlier in this section.

Mass storage permanent datasets

A *mass storage permanent dataset* is available to the system and to other jobs and is maintained across system startups. Permanent datasets are of two types: those created by SAVE requests made by the user or front-end system (user permanent datasets), and input, output, or COS internal datasets (system permanent datasets).

User permanent datasets are maintained for as long as the user or installation desires. They can be protected from unauthorized access by use of permission control words and ownership values.

When a user permanent dataset is accessed via an ACCESS control statement (see section 9 of this manual), it is treated as a local dataset by the job requesting access. However, it still exists as a permanent dataset on the system and can be used by other jobs unless unique access to that dataset was granted. When a user attempts to write to a permanent dataset, the write occurs only when the buffer is at least half full. If any information in an existing permanent dataset is overwritten or if the size of a permanent dataset is changed, an ADJUST should be performed on that dataset (see section 9 of this manual). An ADJUST is performed automatically when a permanent dataset is released.

System permanent datasets relate to particular jobs or reflect the current operational state of COS. A job's *input dataset* is made permanent when the job is received by the Cray Computer System and is deleted when the job terminates. *Output datasets* local to the job can be disposed while the job is running or can be automatically made permanent when the job terminates and then deleted from the Cray Computer System after being sent to the front-end system for processing. An example of a system permanent dataset is the system log.

An *execute-only dataset* is a user permanent mass storage dataset for which all forms of examination and modification by users are prohibited. An execute-only dataset is loaded by the Control Statement Processor (CSP) for execution. It differs in usage from other user permanent datasets in several ways:

- The accessor of the dataset cannot open the dataset for reading or writing.

- While an execute-only dataset is loaded in memory, no DUMPJOB requests are honored.
- The dataset cannot be staged via a DISPOSE request.
- The dataset must be loaded by a dataset name call rather than by the LDR control statement.
- The dataset cannot be dumped via PDSDUMP for archiving purposes.

Because execute-only is a dataset state rather than a permission mode, it is advisable to set, at minimum, a maintenance permission control word to disallow modification or deletion of the dataset.

LOCAL DATASETS

A dataset to which a job has access is a *local dataset*. A local dataset can be temporary or permanent. Permanent datasets are made local with the ACCESS control statement or the ACCESS library subroutine (described in the Library Reference Manual, CRI publication SR-0014). If the dataset referenced is a tape dataset, the device resource must also be specified on the JOB control statement (see section 7 of this manual).

DATASET DISPOSITION CODES

Each dataset is assigned a *disposition code* telling the operating system the disposition to be made of the dataset when the job is terminated or the dataset is released. The disposition code is one of the parameters of the DISPOSE and ASSIGN control statements (see section 8 of this manual).

Each disposition code is a 2-character alphabetic code describing the destination medium of the dataset. The default disposition code for a dataset is SC (scratch) when a dataset is opened, unless the dataset named is one of a group of special names such as $PLOT, $PUNCH, and $OUT. By default, COS assigns the disposition code PR (print) to $OUT when the dataset is created. No DISPOSE statement is required for $OUT; it is automatically routed back to the originating mainframe with a PR (print) disposition.

USER DATASET NAMING CONVENTIONS

The user assigns a symbolic name to each user dataset. This name, the *local dataset name*, is one through seven characters, the first of which must be a through z (in either upper- or lowercase), $, @, or %; remaining characters may also be numeric. A permanent dataset name is less restrictive. A permanent dataset name may contain any printable character. The name must be enclosed in quotes if it contains a character other than upper or lowercase letters, digits, $, @, or %. Do not use characters with the octal codes 000 through 037 or 177 through 377. These are unprintable characters. Refer to the ASCII character set in Appendix C for details. Certain language processors place further restrictions on dataset names.

Most datasets defined by COS are assigned names of the form $*dn*. Since datasets whose names begin with a $ may receive special handling by the system, refrain from using this format when naming datasets.

USER I/O INTERFACES

When using logical I/O, the user is never directly concerned with the actual transfer of data between the devices and the system buffers. Figure 2-6 illustrates the relationship of different levels of user logical I/O interfaces and routines. In this figure, the request levels and routine calls are summarized without going into detail on the movement of data between the system buffers and user program areas. For details on logical I/O, see the Macros and Opdefs Reference Manual, CRI publication SR-0012.

The highest level of user interface is FORTRAN I/O statements; the lowest level is in the form of specially formatted requests called Exchange Processor requests.

FORTRAN statements fall into two categories: formatted/unformatted and buffered. The formatted/unformatted statements result in calls to library routines $RFI through $WUF. If the dataset is blocked, these routines call the logical record I/O routines. The logical record I/O routines perform blocking and deblocking. The logical record I/O routines communicate with COS through the Exchange Processor requests, F$RDC and F$WDC.

If the dataset is unblocked, $RUA or $WUA calls the unblocked dataset routine $RLB or $WLB. These routines do no blocking or unblocking of data. The unblocked I/O routines communicate with the system through the F$RDC and F$WDC Exchange Processor calls.

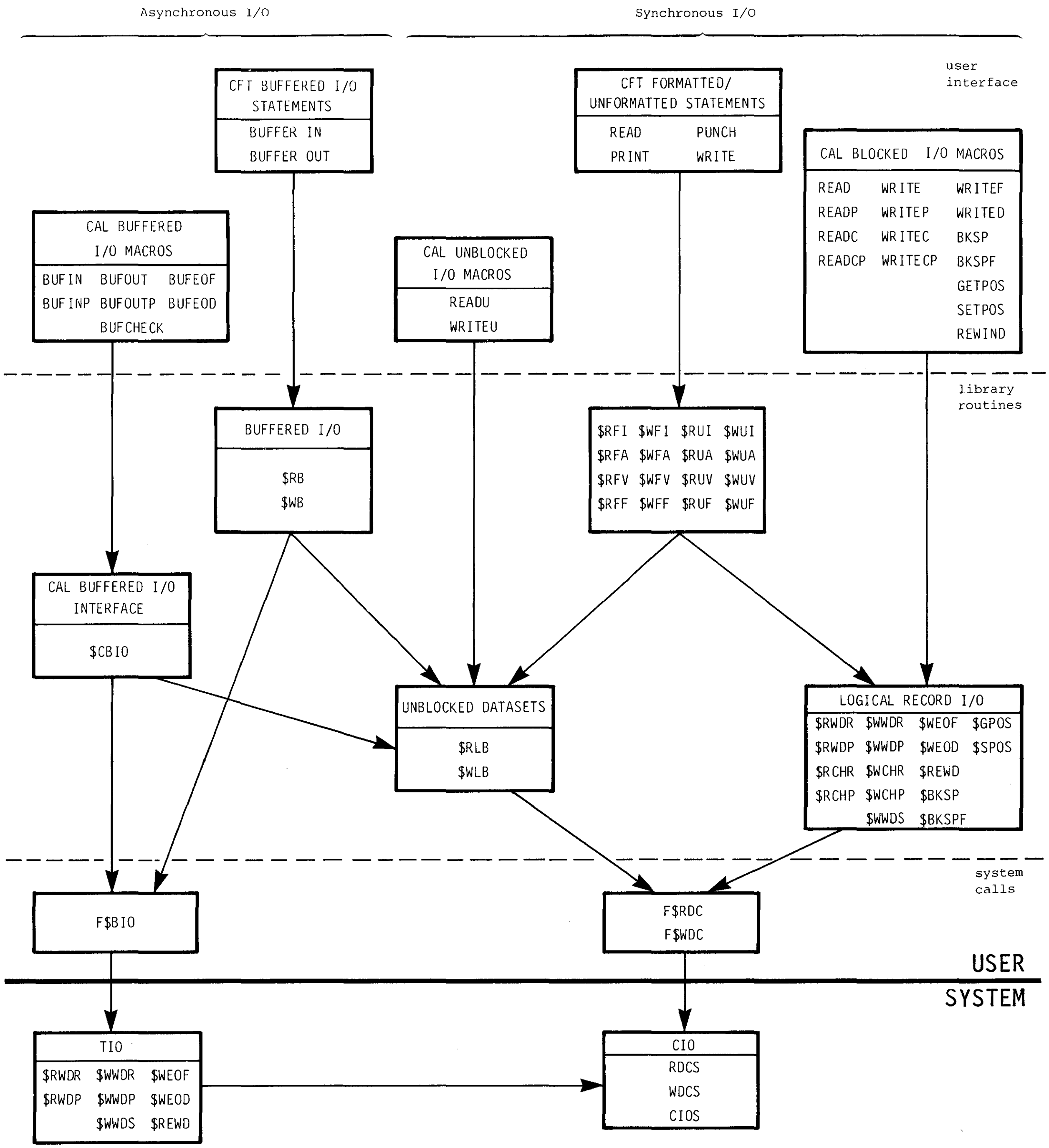

Figure 2-6. Relationship of levels of user I/O

Buffered I/O takes a different path from formatted/unformatted I/O. These routines interface (through an F$BIO Exchange Processor request) to routines in COS that normally perform logical I/O for system tasks. These routines, called Task I/O or TIO, closely resemble the logical record I/O routines. TIO and the logical record I/O routines make similar requests of circular I/O routines in COS although the mechanism for making these requests is different.

Circular I/O routines (CIO) are the focal point for all logical I/O generated by COS. CIO communicates its needs for physical I/O to the Disk Queue Manager or Tape Queue Manager.

A FORTRAN buffered I/O request issued for an unblocked dataset results in the buffered I/O routines calling the unblocked dataset routines $RLB and $WLB, which then process these requests. These requests are processed the same as formatted/unformatted requests except that buffered I/O requests return control to the user after initiating I/O rather than waiting for completion of the I/O request. For a CAL buffered I/O request, $CBIO is called to route the request to either the blocked or unblocked I/O processing routines.

Cray Assembly Language (CAL) I/O macros are described in the Macros and Opdefs Reference Manual, CRI publication SR-0012. Logical record I/O routines and FORTRAN I/O routines are described in the Library Reference Manual, CRI publication SR-0014. See the FORTRAN (CFT) Reference Manual, CRI publication SR-0009, for a description of FORTRAN statements.

COS JOB PROCESSING 3

A *job* is a unit of work submitted to the Cray Computer System. It consists of one or more files of card images contained in a *job deck dataset*. Each job passes through several stages from job entry through job termination.

JOB DECK STRUCTURE

A job originates as a card deck (or its equivalent) at a front-end computer system. Card images in the job deck dataset are organized into one or more files. Figure 3-1 illustrates a typical job deck consisting of a control statement file, a source file, and a data file. (The physical card forms for *end-of-file* and *end-of-data* are defined by the front-end system.)

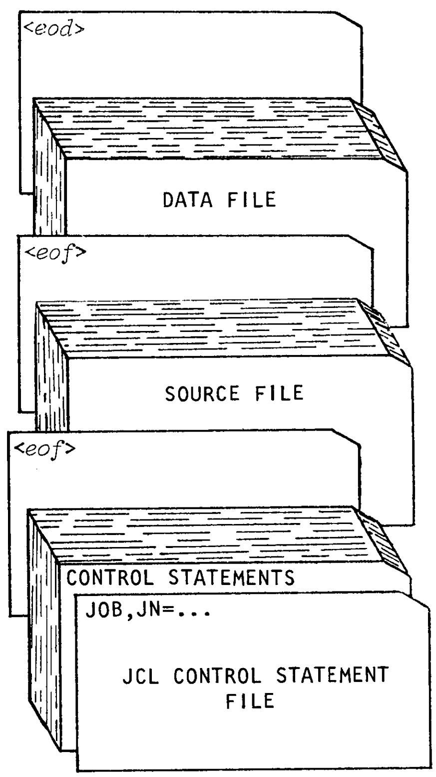

Figure 3-1. Basic job deck

The first (or only) file of the job deck must contain the job control language (JCL) control statements that specify the job processing requirements (JCL is described in section 4 of this manual). Each job begins with a JOB statement, identifying the job to the system. If accounting is mandatory in the user's system, the ACCOUNT statement must immediately follow the JOB statement. All other control statements follow the JOB statement. Control statements can also be grouped into control statement blocks as decribed in section 16 of this manual. The end of the control statement file is designated by an end-of-file record (or an end-of-data record if the job consists of a control statement file only).

Files following the control statement file can contain source code or data. These files are handled according to instructions given in the control statement file.

The final card in a job deck must be an *end-of-data*.

GENERAL DESCRIPTION OF JOB FLOW

A job passes through the following stages from the time it is read by the front-end computer system until it completes:

- Entry
- Initiation
- Advancement
- Termination

JOB ENTRY

A job can enter the system in the form of a dataset submitted from a front-end computer system or a local or remote job entry station. The job is transferred to Cray Computer System mass storage, where it resides until it is scheduled to begin processing. The job input dataset is made permanent until it is deleted at the completion of the job.

JOB INITIATION

The operating system examines the parameters on the JOB control statement to determine the resources needed. When system resources required for initiation are available, the job is initiated (scheduled to begin processing).

Initiation of a job includes preparing a Job Table Area (JTA) and user field, positioning the input dataset for the first job step, and placing the job in a waiting queue for the CPU.

When COS schedules the job for processing, it creates four datasets: $CS, $IN, $OUT, and $LOG.

$CS is a copy of the job's control statement file from $IN and is used only by the system; the user cannot access $CS by name. This dataset is used to read job control statements. The disposition code for $CS is SC (scratch).

$IN is the job input dataset. The job itself can access the input dataset, with read-only permission, by its local name, $IN, or as FORTRAN unit 5.

$OUT is the job output dataset. The job can access this dataset by name or as FORTRAN unit 6. The disposition code for $OUT is PR (print).

The job's logfile ($LOG) contains a history of the job. This dataset is known only to the operating system and is not accessible to the user. User messages can be added to the job's logfile with the MESSAGE system action request macro (see the Macros and Opdefs Reference Manual, CRI publication SR-0012) or the REMARK, REMARK2, or REMARKF subroutines (see the Library Reference Manual, CRI publication SR-0014).

JOB ADVANCEMENT

Job advancement is the processing of a job according to the instructions in a control statement file. Advancement occurs as a normal advance or as an abort advance.

A normal advance causes COS to interpret the next control statement in the job's control statement file. When a job step is multitasked, a job advance deletes all user tasks except the one that causes the advance.

An abort advance occurs if the operating system detects an error or if the user requests that the job abort. Abort advances are described fully under Exit Processing later in this section.

JOB TERMINATION

Output from a job is placed on system mass storage. At completion of a job, the operating system appends $LOG to $OUT and returns $OUT to its originating station. $IN, $CS, and $LOG are released. $OUT is renamed *jn* (from the JN parameter value of the JOB control statement described in section 7 of this manual) and is directed to the output queue for staging to the specified front-end computer system. When the front end has received the entire contents of $OUT, the output dataset is deleted from COS mass storage.

The front-end computer processes $OUT as specified by the dataset disposition code. If, for any reason, $OUT does not exist, $LOG is the only output returned at job termination.

If EXP encounters an error as it attempts to copy $LOG to $OUT, $LOG is disposed on a separate file.

JOB MEMORY MANAGEMENT

Central Memory is a resource that is allocated to jobs by the operating system. A job's memory is composed of several distinct areas. Some of these areas are managed exclusively by the system for the user; others are managed by both the system and the user.

Figure 3-2 illustrates a job in memory. The total job size equals the length of the job's Job Table Area (JTA) plus user field length. The lined area between JCHLM and JCLFT is unused space within the job. This area contains enough memory to guarantee that the job size is always a multiple of 512 words.

INITIAL MEMORY ALLOCATION

When the job initiates it is given sufficient memory for the Control Statement Processor (CSP) to execute. Once the JOB statement is processed, the job is allowed a field length no larger than the amount specified by the MFL parameter on the JOB control statement (see section 7 of this manual).

MODES OF FIELD LENGTH REDUCTION

There are two modes of field length reduction: automatic and user managed.

- Automatic field length reduction mode. When the job is in automatic field length reduction mode, the system automatically increases and decreases the job's field length as the areas within the job increase and decrease. A job initiates in automatic field length reduction mode.

- User-managed field length reduction mode. When the job is in user-managed field length reduction mode, the system continues to increase the job's field length as before, but never automatically decreases it. The job's field length can be decreased only by the user until the job is returned to automatic field length reduction mode.

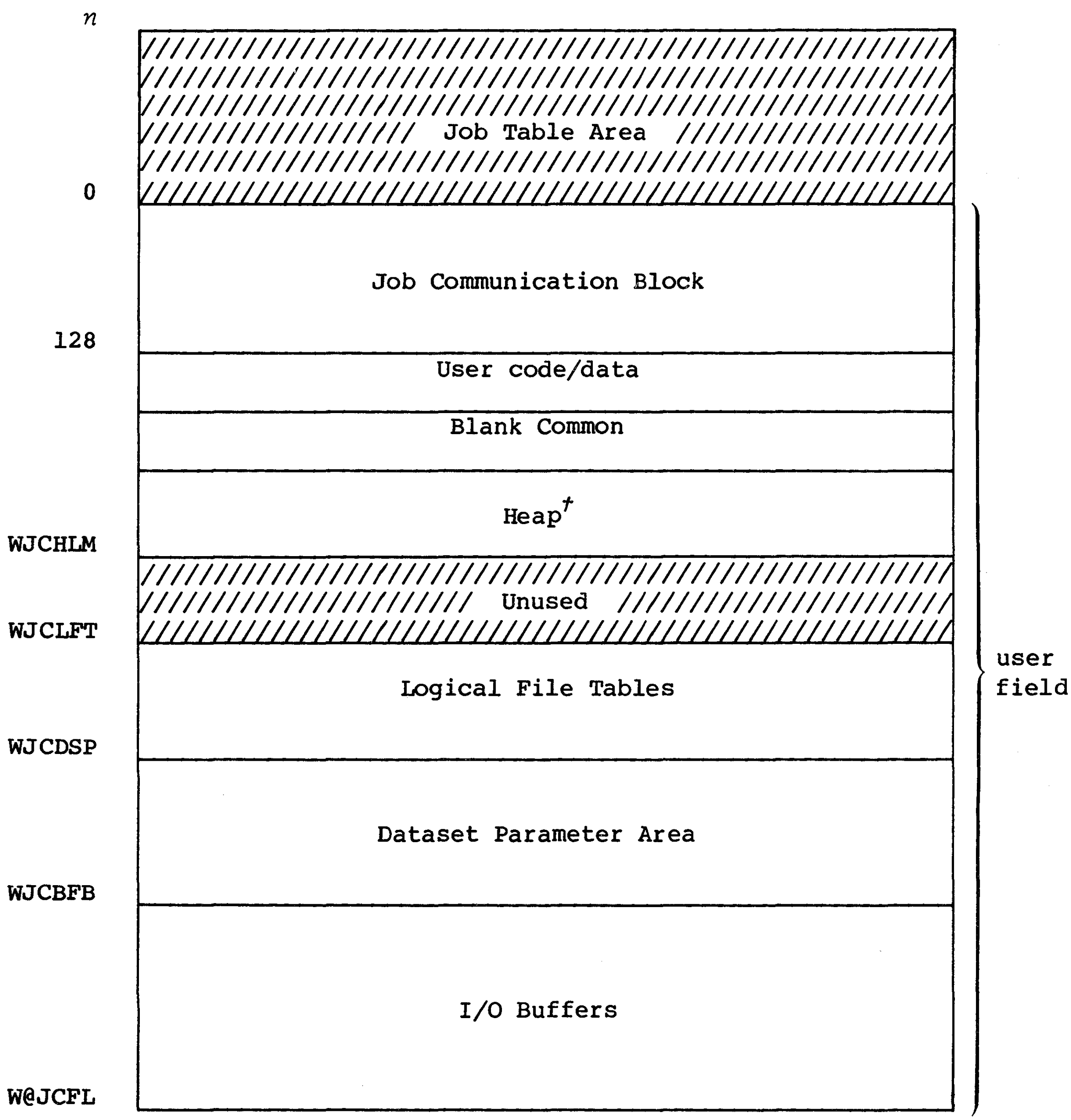

Figure 3-2. User area of memory for a job

† Although the heap follows blank common in the figure, it can optionally precede blank common.

The field length can be reduced at the beginning of each job step and during each job step if the job is in automatic field length reduction mode and any area of the job decreases. Since increases in field length can result in the job's requiring more memory than can be immediately supplied, which causes the job to be delayed until sufficient memory can be given to it, the user may want to manage the job's field length when it is known that the job will undergo frequent short-lived fluctuations in size.

USER MANAGEMENT OF MEMORY

A user can dynamically manage the user code/data area of the job by requesting an increase or decrease of memory at the end of the user code/data area.

A user can manage the field length of the job by requesting a specific field length.

When the user manages the field length of the job, the job is placed in user-managed field length reduction mode for the duration of the job step (next job step when using the MEMORY control statement described in section 9 of this manual).

A user can place the job in user-managed field length reduction mode across job steps by explicitly requesting that mode. The job remains in user-managed field length reduction mode until the user explicitly requests automatic field length reduction mode.

Management by control statement from the run stream

A user can use the MEMORY control statement to manage the job's field length. When the user manages the job's field length, the job will be placed in user-managed field length reduction mode for the duration of the next job step. The MEMORY control statement may also place the job in user-managed field length reduction mode across job steps or return the job to automatic mode.

Management from within a program

From within a program, use of the MEMORY macro or MEMORY routine, respectively, requests user management of the job's user code/data area and field length. When the user manages the job's field length, the job is placed in user-managed field length reduction mode for the duration of the job step. The MEMORY macro or MEMORY routine may also place the job in user-managed field length reduction mode across job steps or return the job to automatic mode.

Management associated with a program

Use of the BC, PAD, and NORED parameters on the LDR control statement (see section 14 of this manual) causes certain memory management to be associated with the binary being loaded. This association is stored with the binary if the binary is saved on a dataset. The management associated can be user code/data area management or field length management and occurs when the binary is loaded for execution. If the field length is being managed, the job is placed in user-managed field length reduction mode for the duration of program execution.

SYSTEM MANAGEMENT OF MEMORY

The system changes appropriate areas of the job's memory when a job initiates certain system actions (that is, advances to the next job step, does I/O, etc.). The Job Table Area, Logical File Tables, and Dataset Parameter Area pictured in figure 3-2 can increase, but will never decrease. The user code/data and buffer areas may both increase and decrease in size. If the job is in automatic field length reduction mode, the system automatically increases and decreases the job's field length when any area in the job increases or decreases. If the job is in user-managed field length reduction mode, the system continues to increase the field length when it needs to, but never automatically decreases the field length.

JOB RERUN

Under certain circumstances, restarting of a job from its beginning may become necessary or desirable. This is referred to as rerunning a job. Conditions causing the system to attempt to rerun a job are:

- Operator command,
- Uncorrectable memory error,
- Uncorrectable error reading the mass storage image of a job, and
- System restart.

A user job may perform certain functions that normally make its rerunning impossible. The functions render a job nonrerunnable because they produce results that might cause the job to run differently if it were rerun. These functions include:

- Writing to a permanent dataset

- Saving, deleting, adjusting, or modifying a permanent dataset

- Acquiring a dataset from a front-end system

Ordinarily, when a job becomes nonrerunnable, it remains so. However, the user may declare that the job is rerunnable. The user should do this only when changes in job results due to execution of nonrerunnable functions are acceptable. COS never makes a job rerunnable automatically.

The user can also override system monitoring of job rerunnability, regardless of what functions the job performs. This ordinarily is done only if the job is structured to run correctly regardless of whether nonrerunnable functions are performed.

EXIT PROCESSING

When an error condition is detected by COS or when the user requests a job step abort, COS checks to see if the condition is to be reprieved (Reprieve Processing is described in the next subsection). If no reprieve occurs, exit processing occurs. Generally, when a job step abort occurs, the current job step is immediately abandoned and control statements are skipped until the next eligible EXIT statement is encountered (EXIT is described in section 7). Normal job advancement occurs with the EXIT statement that is found. If no eligible EXIT statement is found, the job is terminated.

EXIT statements that are within control statement blocks (iterative, conditional, or in-line procedure) that have not yet been invoked are ignored during the search for the next eligible EXIT statement.

If the block currently being processed is a conditional block (see section 16), only the group of control statements preceding the next conditional statement in the block is searched for an eligible EXIT statement; if none is found, the search continues with the first statement following the conditional block. For example, in the following sample control statement sequence, an abort advance occurs at the control statement THIS IS A JOB STEP ABORT CONDITION because it does not begin with a valid verb. Control statement interpretation resumes with the control statement: *. RESUME HERE. The EXIT statements that are included in the conditional block are ignored because they reside in blocks that are not executed.

```
     .
     .
     .
     SET,J1=0.
     IF(J1.EQ.0)
       .
       .
       .
       THIS IS A JOB STEP ABORT CONDITION.
     ELSEIF (J1.EQ.1)
         .
         .
         .
         EXIT.
     ELSE.
         EXIT.
     ENDIF.
         .
         .
         .
     EXIT.
     *.  RESUME HERE
         .
         .
```

Exit processing is not performed for interactive jobs except inside an invoked procedure. After a job step abort occurs, the user is simply prompted for the next control statement.

REPRIEVE PROCESSING

Normally, when a job step abort error occurs, exit processing begins (see the previous section for a full description of exit processing). Reprieve processing, however, allows a user program to attempt recovery from many of the job step abort errors or to perform clean-up functions before continuing with the abort.

Reprieve processing can also be used during the normal termination of a job step. In this case, control transfers to the user's reprieve code instead of to the next normal job step.

Two types of error conditions are related to a job step: nonfatal and fatal.

- Nonfatal error conditions are those which can be reprieved any number of times per job step by the user.

- Fatal error conditions can be reprieved only once for each type per job step.

See Appendix D for a listing of all fatal and nonfatal error conditions.

When requesting reprieve processing, the user selects the error conditions to be reprieved by setting a mask in the SETRPV subroutine or macro call. If a selected error condition occurs during job processing, the user's current job step maintains control. The user's exchange package, vector mask register, error code, and error class are saved and control passes to the user's reprieve code.

INTERACTIVE JOB PROCESSING

An interactive job dataset has interleaved control statements, program or utility input, and program or utility output. In an interactive job, the control statement file ($CS), standard input dataset ($IN), standard output dataset ($OUT), and logfile ($LOG) are all defined by the system to be interactive datasets. See section 2 for more information on interactive datasets.

Each job step of an interactive job is initiated with a control statement. Control statements can be either part of a procedure invocation or entered directly from the interactive terminal. After each control statement is received by COS, input to the job step can be entered from the terminal and output and logfile information is returned to the terminal. When the current job step is complete, normal job advancement occurs and COS prompts for the next control statement (or reads it from the invoked procedure file). Exit processing (see section 3) is never performed on an interactive job except within a procedure invocation.

Whenever a program or utility executing as part of an interactive job requests to read from the standard input dataset, the interactive user is prompted to enter data one record at a time. Likewise any data written to $OUT, the standard output dataset, is sent to the interactive terminal. User logfile messages are also sent to the interactive terminal.

JOB LOGFILE AND ACCOUNTING INFORMATION

For each job run, the system produces a logfile--an abbreviated history of the progress of the job through the system. The logfile for a job appears at the end of the job output. Each job control statement is listed sequentially, followed by any messages associated with the job step. Clock time, accumulated CPU time, and COS information are also given for each job step. A logfile usually consists of the items illustrated in figure 3-3. Item 6 illustrates the accounting information given to the user.

```
                 3
14:57:06    0.0000   CSP    ------------------------------------------------------------------------
14:57:06    0.0000   CSP    03/07/83 - The current COS on SN27 is 03/03. This system was brought up
14:57:06    0.0000   CSP         7    at 1000, 03/07.
14:57:06    0.0000   CSP
14:57:06    0.0000   CSP    ------------------------------------------------------------------------
14:57:06    0.0000   CSP     1   CRAY-1 SERIAL 27/4         CRI - MENDOTA HEIGHTS, MINN. 03/23/83
14:57:06    0.0000   CSP
14:57:06    0.0001   CSP     2   CRAY OPERATING SYSTEM           COS 1.12  ASSEMBLY DATE 03/03/83
14:57:06    0.0001   CSP
14:57:06    0.0001   CSP
14:57:06    0.0001   CSP    JOB,JN=SAMPJOB,US=PROJECT2013,MFL=28000,T=1.
14:57:06    0.0010   CSP    ACCOUNT,AC=.
14:57:07    0.0017   EXP    *
14:57:07    0.0017   EXP  4 *  GENERATE A PERMANENT DATASET
14:57:07    0.0017   EXP    *
14:57:07    0.0022   CSP    COPYF(O=PERMDS)
14:57:07    0.0024   USER   FT048 - COPY OF        9 RECORDS       1 FILES COMPLETED
14:57:07    0.0029   CSP    COPYF,O=PERMDS.
14:57:07    0.0040   USER   FT048 - COPY OF       72 RECORDS       1 FILES COMPLETED
14:57:08    0.0043   CSP    SAVE(DN=PERMDS,ID=P2013)
14:57:08    0.0043   PDM  5 PD000 - PDN = PERMDS          ID = P2013     ED =     1  US = PROJECT2013
14:57:08    0.0043   PDM    PD000 - SAVE    COMPLETE
14:57:08    0.0043   CSP    EXIT.
14:57:08    0.0044   CSP    END OF JOB
14:57:08    0.0044   CSP
14:57:08    0.0044   CSP
14:57:08    0.0046   USER   6  JOB NAME -                         SAMPJOB
14:57:08    0.0046   USER      USER NUMBER -                      PROJECT2013
14:57:08    0.0047   USER      TIME EXECUTING IN CPU -            0000:00:00.0046
14:57:08    0.0047   USER      TIME WAITING TO EXECUTE -          0000:00:00.2999
14:57:08    0.0047   USER      TIME WAITING FOR I/O -             0000:00:01.6256
14:57:08    0.0047   USER      TIME WAITING IN INPUT QUEUE -      0000:00:00.0624
14:57:08    0.0048   USER      MEMORY * CPU TIME (MWDS*SEC) -            0.11527
14:57:08    0.0048   USER      MEMORY * I/O WAIT TIME (MWDS*SEC) -      47.48925
14:57:08    0.0048   USER      MINIMUM JOB SIZE (WORDS) -           13312
14:57:08    0.0048   USER      MAXIMUM JOB SIZE (WORDS) -           30208
14:57:08    0.0048   USER      MINIMUM FL (WORDS) -                 10240
14:57:08    0.0048   USER      MAXIMUM FL (WORDS) -                 27136
14:57:08    0.0048   USER      MINIMUM JTA (WORDS) -                 3072
14:57:08    0.0049   USER      MAXIMUM JTA (WORDS) -                 3072
14:57:08    0.0049   USER      DISK SECTORS MOVED -                    96
14:57:08    0.0049   USER      USER I/O REQUESTS -                     18
14:57:08    0.0049   USER      OPEN CALLS -                            11
14:57:08    0.0049   USER      CLOSE CALLS -                            9
14:57:08    0.0049   USER      MEMORY RESIDENT DATASETS -               0
14:57:08    0.0049   USER      TEMPORARY DATASET SECTORS USED -         0
14:57:08    0.0049   USER      PERMANENT DATASET SECTORS ACCESSED -    20
14:57:08    0.0049   USER      PERMANENT DATASET SECTORS SAVED -        1
14:57:08    0.0049   USER      SECTORS RECEIVED FROM FRONT END -        0
14:57:08    0.0049   USER      SECTORS QUEUED TO FRONT END -            0
```

Figure 3-3. Example of a job logfile

1 First header line: Installation-defined message, usually identifying the site and date the job was run.

2 Second header line: Installation-defined message, usually identifying the operating system, its current revision level, and the date of the last revision.

(3) Columns: The leftmost column identifies the wallclock time for each job step and the middle column identifies the accumulated CPU time for the job. The rightmost column identifies a system module or the user as the originator of the message. All times are in decimal. Entries commonly noted include the following:

CSP	Control Statement Processor
PDM	Permanent Dataset Manager
EXP	Exchange Processor
ABORT	Abort Message
USER	Program in user field

(4) Control statements: Control statements are listed in the logfile as they are processed unless requested otherwise with the ECHO statement described in section 6 of this manual. When the job terminates, the last control statement processed that may be echoed is the last control statement printed. Control statements are not listed if the JCL message class (see the ECHO control statement) is disabled.

(5) Logfile messages: Any messages related to control statement processing are shown below the statement.

(6) Accounting information: When a job reaches completion, COS writes a summary of basic accounting data onto the logfile for the job. All times given are in hours, minutes, and seconds (to the nearest ten-thousandth of a second). The following accounting information is provided (in decimal):

- Job name and user number
- CPU time used by the job and by each job task in a multitasked job step
- Time waiting to execute, for the job and for each job task in a multitasked job step; includes time waiting for the CPU, memory, operator suspension, and recovery.
- Time waiting for I/O, for the job and for each job task in a multitasked job step
- Time waiting in input queue
- Memory usage based on the execution and I/O wait time in million word-seconds
- Minimum and maximum job size including Job Table Area (JTA) (words)
- Minimum and maximum field length used (words)

- Minimum and maximum JTA used (words)
- Number of 512-word disk blocks (sectors) moved
- Number of user I/O requests made by the job
- Open and close calls
- Memory-resident datasets
- Number of 512-word disk blocks (sectors) used for temporary datasets
- Number of 512-word disk blocks (sectors) accessed and saved for permanent datasets
- Number of 512-word disk blocks (sectors) received from and queued to the front end
- Number of tape devices reserved; message issued only if magnetic tape datasets have been processed.
- Number of tape volumes mounted; message issued only if magnetic tape datasets have been processed.
- Amount of tape data moved, expressed as a multiple of 512 words; message issued only if magnetic tape datasets have been processed. Each disk sector consists of 512 words, and in COS blocked format each block consists of 512 words.
- Number of tape blocks moved; message issued only if magnetic tape datasets have been processed.

7 System Bulletin: The system bulletin allows the installation to print messages in the logfile, usually about the status of the system environment. It is an installation-maintained message dataset.

JOB CONTROL LANGUAGE 4

The job control language of the Cray Operating System (COS) allows the user to present a job to the Cray Computer System, define and control execution of programs, and manipulate datasets.

The job control language is composed of *control statements* with each control statement containing information for a job step. COS initially creates a *control statement dataset*, $CS, to hold job control statements. Additional control statement datasets can be created through procedure definition or the CALL control statement (see section 6).

The syntax of a control statement is:

verb	sep_1	$param_1$	sep_2	$param_2$	...	sep_n	$param_n$	*term*	*comments*

All control statements must adhere to a set of general syntax rules. Every control statement consists of a *verb* and a terminator (*term*) as a minimum, except for the comment control statement (*) which does not require a terminator. Additionally, most control statements require parameters ($param_i$) and separators (sep_i) between the verb and the terminator. The maximum number of parameters (zero, one, or more) depends on the verb.

The continuation separator (the caret symbol) allows a control statement to consist of more than one line image (80 characters). The JOB, ACCOUNT, DUMPJOB, EXIT, and comment control statements cannot be continued. All other control statements can have any number of continuation card images, subject to restriction by the verb. A caret occurring within a literal string has no special significance.

A *comment* is an optional annotation to a control statement and can be a string of any ASCII graphic characters. The comment follows the line image terminator. The control statement interpreter ignores comments. All comments appear in the logfile unless suppressed by the ECHO control statement.

Blanks are ignored unless they are embedded in a literal string. Blanks cannot precede the verb on the JOB control statement.

SYNTAX VIOLATIONS

COS notes syntax violations in the system and user logfiles. If the JOB control statement is in error, processing of the job terminates immediately. If accounting is mandatory, ACCOUNT statement errors also cause job termination. All other syntax errors cause a *job step abort* condition, which causes the system to search for an EXIT control statement. A successful search resumes control statement processing with the job step following EXIT. If no such job step exists or if an EXIT statement is not found, the job is terminated. Job step abort can also direct control to a user-specified routine (see exit processing and reprieve processing in section 3).

VERBS

A *control statement verb* is the first nonblank field of a control statement specifying the action to be taken by COS during control statement processing. COS recognizes three types of control statement verbs: *system verbs*, *dataset name verbs* (*local* and *system*), and *library-defined verbs*. A control statement verb cannot be continued across a card boundary.

When COS encounters a verb in a control statement file, it searches for a match to that verb in the following order:

1. System verbs
2. Local dataset name verbs
3. Library-defined verbs
4. System dataset name verbs

COS first searches the list of system verbs for a match. If the verb is not a system verb, COS searches for a local dataset name that might match the verb. If the verb is not the name of a local dataset, COS searches each library in the library searchlist for a match. If it does not find a library entry that matches the verb, it searches the System Directory Table (SDR) for a matching system dataset name. If a match for the verb is not found under any of these categories, COS issues a control statement error and aborts the job step.

SYSTEM VERBS

A system verb consists of an alphabetic character which can be followed by one through seven alphanumeric characters.† The verb requests that COS perform the indicated function. The system verbs are:

*	DISPOSE	EXIT	LIBRARY	PERMIT	ROLLJOB
ACCESS	ECHO	EXITIF	LOOP	PRINT	SAVE
ACQUIRE	ELSE	EXITLOOP	MEMORY	PROC	SET
ADJUST	ELSEIF	FETCH	MODE	RELEASE	SIMABORT
ASSIGN	ENDIF	IF	MODIFY	RERUN	SUBMIT
CALL	ENDLOOP	IOAREA	NORERUN	RETURN	SWITCH
DELETE	ENDPROC	JOB	OPTION	REWIND	

The SIMABORT control statement is described in the COS Simulator (CSIM) Reference Manual, publication SR-0073.

LOCAL DATASET NAME VERBS

A verb that is the name of a local dataset consists of an alphabetic character followed by one through six alphanumeric characters.† This verb requests that COS load and execute an absolute binary program from the first record of the named dataset. If the user job has a dataset with the indicated name, COS loads and executes the program from that dataset.

LIBRARY-DEFINED VERBS

A library-defined verb consists of one through eight characters. The library-defined verb is either a program or procedure definition (see section 16 of this manual) residing in a library that is a part of the current *library searchlist*. (The library searchlist defines the library and the order in which the libraries are searched by COS. This order can be specified with the LIBRARY statement described in section 7.) A program in a library is an absolute binary program to be loaded and executed. A procedure definition is a group of control statements and/or data to be processed (see section 16).

---

† Alphabetic characters include $, %, @, and the letters A through Z. Alphanumeric characters include all the alphabetic characters and the digits 0 through 9.

SYSTEM DATASET NAME VERBS

COS searches for a verb that is the name of a system-defined dataset in the System Directory Table (SDR). A system-defined dataset name verb consists of an alphabetic character which can be followed by one through six alphanumeric characters.† The System Directory Table is a list of common language processors and utilities known to the system and made available to users at startup. The name of the program (for example, CAL, CFT, or DUMP) is also the name of the dataset containing the absolute binary of the program. The exact list of system dataset name verbs is site-dependent.

SEPARATORS

A *separator* is a character used as a delimiter in a control statement. It separates the verb from the first parameter, separates parameters from one another, delimits subparameters, terminates verbs and parameters, and separates a keyword from its value in parameters having keyword form.

The control statement separators allowed by COS are given in table 4-1.

PARAMETERS

A *parameter* is a control statement argument, whose exact requirements are defined by the control statement verb. Parameters are used in control statements to specify information to be used by the verb-defined process. Parameters that can be used with COS control statements are either *positional* or *keyword*. For certain verbs, a parameter value can be an expression. Detailed information on the use of expressions is presented later in this section. Parameters are separated by commas.

POSITIONAL PARAMETERS

A positional parameter has a precise position relative to the separators in the control statement. Even a null positional parameter must be delimited from the control statement verb or other parameters by a separator.

---

† Alphabetic characters include $, %, @, and the 26 uppercase letters A through Z. Alphanumeric characters include all the alphabetic characters and the digits 0 through 9.

Table 4-1. Control statement separators

Function	Character	Examples
Initial separator (comma or open parenthesis)† - Separates the verb from the first parameter	, (	*VERB,parameter.* *VERB(parameter)*
Statement terminator (period if initial separator is comma; close parenthesis if initial separator is open parenthesis)† - Signifies end of control statement	.)	*VERB.* *VERB,parameter.* *VERB(parameter)*
Parameter separator (comma) - Indicates the end of one parameter and the beginning of the next	,	*VERB(parameter,parameter)*
Equivalence separator (equal sign) - Delimits a parameter keyword from the first parameter value for that keyword. Adjacent equivalence separators are illegal.	=	*VERB(keyword=value)*
Concatenation separator (colon) - Separates multiple parameter values from each other	:	*VERB(keyword=*$value_1$*:*$value_2$*)*
Continuation character (caret) - Indicates that the control statement consists of more than one 80-character card; may appear anywhere after the initial separator	^	*VERB(...parameters...^ parameters)*
Literal string delimiters (apostrophes)†† - Identify the beginning and end of a literal string	'...'	*VERB(keyword='string')*
Parenthesis delimiters (open and close parentheses) - Indicate a group of characters to be treated as one value	(...)	*VERB(keyword=(value:value)*

† By convention in this manual, the comma and period are used as initial and terminator separators for all control statements except or the JCL block control statements (procedure definition, iterative, and conditional) where paired parentheses are conventional.

†† See section 16 for additional information on strings and string delimiters.

The formats for a positional parameter follow:

```
value

value1:value2:...:valuen
```

Each $value_i$ is a string of alphanumeric characters, a literal string, or a null string. All positional parameters are required to be represented by at least one *value*, although the value can be null. Rules for strings are given in section 16.

Examples of positional parameters:

...,ABCDE,...	Parameter value is ABCDE.
...,,...	The adjacent parameter separators indicate a null positional parameter.
...,P1:P2:P3,...	The parameter consists of multiple values.
VERB() or VERB,.	Positional parameter 1 is null

KEYWORD PARAMETERS

A keyword parameter is identified by its form rather than by its position in the control statement. The keyword is a string of one to eight alphanumeric characters uniquely identifying the parameter. Parameters of this type can occur in any order but must be placed after all of the positional parameters for the control statement, or they can be omitted.

The formats of keyword parameters are:

```
keyword

keyword=value

keyword=value1:value2:...:valuen
```

keyword is an alphanumeric string that depends on the requirements of the verb, and $value_i$ is the value associated with the keyword. A keyword parameter can occur anywhere in the control statement after all positional parameters are specified. Whether or not a keyword parameter is required depends on the verb's requirements. If the keyword is not included in the control statement, a default value can be assigned.

Examples of keyword parameters:

...,DN=FILE1,...	Parameter consists of keyword and value.
...,UQ,...	Parameter consists of keyword only.
...,DN=FILE1:FILE2:FILE3,...	Parameter consists of keyword and list of values.
...,DN=,...	Null parameter value, as if omitted from the statement
...,DN=A:::B,...	A, B, and two null parameter values are listed.

The parameter associated with a keyword may be defined as a secure parameter. Every secure parameter is edited out of the statement before it is echoed to the user logfile. When a keyword is secure, all that appears in the user's logfile is the keyword and the = sign, followed by the next delimiter. Secure parameters are defined when calling GETPARAM as described in the Library Reference Manual, CRI publication SR-0014.

PARAMETER INTERPRETATION

The decoding (cracking) of control statement parameters is normally performed by the routines $CCS and GETPARAM, as described in the Library Reference Manual, CRI publication SR-0014. Parameter interpretation is performed by the particular program or utility that calls $CCS or GETPARAM.

CONVENTIONS

The following conventions are used in this manual.

Convention	Description
Italics	Define generic terms representing the words or symbols to be supplied by the user
[ ] Brackets	Enclose optional portions of a command format
{ } Braces	Enclose alternate choices, one of which must be used

LIBRARIES 5

Job control statements, programs, and compiled subprograms are maintained in libraries. The following types of libraries are available on the Cray Operating system:

- Procedure libraries
- Program libraries
- Object code libraries

The CALL and LIBRARY control statements (see section 7 of this manual) refer to procedure libraries; UPDATE (see the UPDATE Reference Manual, CRI publication SR-0013) maintains program libraries; BUILD (see section 15 of this manual) maintains object code and procedure libraries. The LIB and NOLIB parameters of the LDR control statement (see section 14) refer to object code.

PROCEDURE LIBRARY

A *procedure library* is created by the in-line procedure definition process described in section 16 of this manual. After creation, procedure libraries are made available for use by the LIBRARY control statement (see section 7).

A procedure library is made up of procedures which are a sequence of control statements and/or data saved for processing at a later time. Procedures are described in section 16 of this manual.

PROGRAM LIBRARY

A *program library* is a means of maintaining programs and other data on datasets. These datasets are created and maintained by the UPDATE utility described in the UPDATE Reference Manual, CRI publication SR-0013. A program library (PL) consists of one or more specially formatted card image decks, each separated by an end-of-file record. These decks can be programs, portions of programs, input data for programs, or even job control statements. See the UPDATE Reference Manual for full information on using program libraries.

OBJECT CODE LIBRARIES

Object code libraries are termed *library datasets* or simply *libraries*. A *library dataset* is a dataset containing a program file followed by a directory file. Library datasets are designed primarily to provide the Relocatable Loader (see section 14) with a means of rapidly locating and accessing program modules. Library datasets are created and maintained by the BUILD utility as described in section 15 of this manual. Any library dataset can be inspected and described by ITEMIZE. See section 13 for more information on ITEMIZE.

JOB CONTROL STATEMENTS 6

Job control statements perform the following functions:

- Identify a job to the system
- Define operating characteristics for the job
- Manipulate datasets
- Call for the loading and execution of user programs
- Call COS programs that perform utility functions for the user
- Define and manipulate other control statements

The first file of a job dataset contains control statements that are read, interpreted, and processed one at a time. The sequential processing of control statements determines the *job flow* through the operating system. See section 3 for a general description of job flow. Sequential processing of control statements can be altered by exit or reprieve processing, or by control statement structures described in section 16.

Information on the general syntax rules and conventions for control statements is presented in section 4. Sections 6 through 15 describe COS control statements and give example in some cases. The control statements are described in the following categories:

- Job definition
- Dataset definition and control
- Permanent dataset management
- Dataset staging control
- Permanent dataset utilities
- Local dataset utilities
- Analytical aids
- Executable program creation
- Object library management

JOB DEFINITION

Several control statements allow the user to specify job processing requirements. Control statements defining a job and its operating characteristics to the operating system include the following.

Verb	Function
JOB	Introduces the job to the operating system and defines characteristics such as size, time limit, and priority levels
MODE	Sets or clears mode bits in the job's Exchange Package
EXIT	Indicates the point in a series of control statements at which processing of control statements resumes following a job step abort from a program or indicates the end of control statement processing
MEMORY	Requests a new field length and/or mode of field length reduction
SWITCH	Turns on or turn off pseudo sense switches
*	Annotates control statements with comments
RERUN, NORERUN	Controls job rerunnability
IOAREA,	Denies or allows access to the job's I/O area, the upper (high-address) portion of user memory that contains tables and buffers managed by the system I/O library routines
CALL, RETURN	Allows the use of alternate control statement files
ACCOUNT	Validates the job's account number, user number, and optional passwords
CHARGES	Obtains partial or total resource reporting for a job
ROLLJOB	Protects a job by writing it to disk
SET	Changes the value of a job control language (JCL) symbolic variable
ECHO	Controls types of messages written to the job's logfile

Verb	Function
LIBRARY	Specifies the datasets to be searched, when looking for defined procedures, during job processing. LIBRARY also specifies the order in which to perform the search.
OPTION	Specifies user-defined options, such as the format of the job's listing and the amount of dataset accounting statistics produced

Job definition and control statements are fully described in section 7.

DATASET DEFINITION AND CONTROL

Datasets can be defined and managed by the user with the following dataset control statements: ASSIGN, ACCESS, and RELEASE.

Verb	Function
ASSIGN	Defines characteristics for datasets, such as the amount of user memory to allocate for the dataset's I/O buffer. ASSIGN also can be used to create a mass storage dataset. ACCESS must first be used to create a tape dataset.
RELEASE	Relinquishes access to the named dataset for the job

ASSIGN and RELEASE are fully defined in section 8. ACCESS is described later in this section under Permanent Dataset Management because it is primarily used in managing permanent datasets.

PERMANENT DATASET MANAGEMENT

Control statements for managing permanent datasets provide for creating, protecting, and accessing datasets assigned permanently to mass storage or magnetic tape. Such datasets cannot be destroyed by normal system activity or engineering maintenance.

Front-end computer systems cannot directly affect Cray-resident permanent datasets, since permanent dataset management is handled entirely by COS. However, permanent magnetic tape dataset management can optionally be coordinated with a front-end computer system.

Users can manage user permanent datasets only; system permanent datasets cannot be managed (modified or deleted) by the user. (See section 2 for a description of the types of datasets.)

The control statements available for user permanent mass storage and magnetic tape dataset management are shown in table 6-1. Actual processing of these requests depends upon the medium on which the dataset resides. Mass storage datasets are controlled by the COS system task called the Permanent Dataset Manager (PDM). Magnetic tape datasets are controlled by a system task called the Tape Queue Manager (TQM). Both of these system tasks (PDM and TQM) have mechanisms for retaining the characteristic information about the dataset. Information for mass storage datasets is retained in the Central Memory-resident Dataset Catalog (DSC). Magnetic tape datasets can have characteristic information retained on a front-end computer system.

The permanent dataset management control statements are fully described in section 9.

MASS STORAGE DATASET ATTRIBUTES

Every mass storage permanent dataset has several *attributes* associated with it. These attributes are:

- Read, write, and maintenance permission control words,
- Public access mode,
- Public access tracking,
- Permits,
- Text, and
- Notes

Permission control words

A *permission control word* is a password that must be supplied to gain access to a particular permanent dataset. Permanent datasets are not required to have a permission control word, but if a permission control word is specified for the mode of dataset access desired (read, write, maintenance), the control word must be specified to gain access to the named dataset. If more than one mode of access is desired (for example, both read and write), all appropriate control words must be supplied.

Table 6-1. Permanent dataset management control statements for each medium

Verb	Mass storage	Magnetic tape
SAVE	Enters a dataset's identification and location in a system-maintained Dataset Catalog. Datasets recorded in the Dataset Catalog via a user SAVE request are user permanent datasets and are recoverable at deadstart.	Supplies to a front-end computer system the characteristic information about a dataset for its retention
ACCESS	Assigns (makes local) a user permanent dataset to the requesting job, with the requested and/or allowable modes (execute, read, write, maintenance)	Assigns an existing tape dataset to the job or defines a NEW-type tape dataset that will be created by the job. Also optionally, defines the front-end computer system that will be the central point for servicing that dataset.
DELETE	Removes the definition of a user permanent dataset from the Dataset Catalog (DSC). It is possible to delete a dataset's contents and have its attributes retained by the system.	Requests the front-end computer system servicing the dataset to remove (delete) any information concerning the dataset
MODIFY	Changes the characteristic information for an existing user permanent dataset	Not applicable
ADJUST	Records the change in any of the size or allocation information for a dataset that might have contracted or expanded	Not applicable
PERMIT	Explicitly grants or denies specified users or groups of users access to a permanent dataset	Not applicable

Public access mode attribute

If all users are to be allowed some kind of access to a permanent dataset, that dataset must have a *public access mode* defined. The public access mode is the type of access, as a minimum, all users can have to the permanent dataset. Users can be allowed read, write, and/or maintenance mode access to the dataset. Users can be restricted to only executing the dataset; the public access mode can alternatively be NONE, signifying that public access is not permitted.

Public access tracking attribute

Public access tracking is a facility that can be turned on or off. A record can be kept of every user who accesses a public dataset. See Dataset Use Tracking later in this section for more details on the public access tracking mechanism.

Permits attribute

User permanent mass storage datasets can have a list of alternate users of the dataset and in what mode or modes each alternate user can access the dataset. Each element of the list is known as a *permit* and names a specific alternate user and that user's allowed mode of dataset access. Permits are described more fully under Access Mode later in this section.

Text attribute

text is a character string to be passed to a front-end computer system when requesting transfer of the dataset to or from Cray mass storage. Text is more fully described under Dataset Staging Control later in this section.

Notes attribute

notes is a string of up to 480 characters associated with a permanent dataset. There is no restriction on what *notes* contains. When *notes* is listed using the AUDIT utility (see Permanent Dataset Utilities later in this section), the caret symbol is interpreted as an end-of-line signal and AUDIT advances to a new line when listing the dataset *notes*. *notes* can contain such information as dataset structure, usage instructions, or history. For example, if several versions of a program exist as different permanent datasets, the *notes* could identify the purpose, difference, and origin of each dataset.

ESTABLISHING ATTRIBUTES FOR MASS STORAGE DATASETS

Mass storage permanent dataset attributes are established at dataset creation time, though they can be later modified (or added to in the case of permits). Attribute establishment depends on whether a dataset with the same name (PDN), additional identification (ID), and ownership already exists.

Supplying the entire set of attributes every time a new permanent dataset is created, that is, when no permanent dataset with the same PDN, ID, and ownership currently exists, can become quite tedious, especially if a long list of permits must be established. Instead, the dataset creator can supply an *attributes dataset.*

Existing permanent dataset

If a permanent dataset with the requested PDN, ID, and ownership already exists, the current dataset's permission control words, public access mode, public access tracking, and permit list are set to the corresponding attributes of the permanent dataset with the highest existing edition number (ED) and identical PDN, ID, and ownership.

The text attribute is also copied from the highest existing edition unless otherwise specified; the notes attribute is not copied.

The discussion of creating a new edition of an existing permanent dataset applies to datasets created by SAVE or PDSLOAD (see Permanent Dataset Utilities later in this section for information on PDSLOAD). If MODIFY is used to create a new edition of an existing dataset (by changing the PDN or ID), any dataset attributes not explicitly modified remain unchanged. Thus, it is possible, though not recommended, for different permanent datasets with the same PDN, ID, and ownership to have different attributes.

New permanent dataset

Using SAVE or ACQUIRE when no permanent dataset currently exists with the same PDN, ID, and ownership causes a new permanent dataset to be created.

All permanent dataset attributes can be established for a new permanent dataset; no attribute is associated with any other dataset. For example, if the new permanent dataset is to have a read permission control word, then the control word must be supplied. If a list of permits is needed, then the list must be supplied. Establishing an attributes dataset provides a convenient way of supplying a list of permits described in the following subsection.

Attributes dataset

An *attributes dataset* is an existing permanent mass storage dataset from which any (or all) permanent dataset attributes can be copied. The actual dataset content is ignored; the attributes are copied from the dataset's catalog entry. The attributes dataset can even be partially deleted (see Dataset Staging Control later in this section for a discussion of partial dataset deletion). The attributes dataset must be local to the job referencing it.

The attributes dataset is referenced with the ADN parameter on the SAVE or ACQUIRE control statement. When the attributes dataset is referenced, all desired attributes (such as permission control words and the public access mode) are copied from the attributes dataset and used in establishing the attributes of the current dataset. Any attribute explicitly specified on the SAVE or ACQUIRE control statement is used instead of the attributes dataset's attribute. Examples of attribute dataset use are included at the end of section 9.

An attributes dataset can also be used with the PERMIT control statement, although it is used slightly differently. When an attributes dataset is used with PERMIT, the entire permit list (but no other attribute) is copied from the attributes dataset and added to the permit list established (or being established) for the current dataset.

For example, suppose the same permit list is being used for several different datasets. A single permanent dataset can be created and the list of permits established. Then whenever a new dataset is created, the original dataset can be accessed and used as an attributes dataset. The new dataset creator need not even know what permits are being established.

PROTECTING AND ACCESSING MASS STORAGE DATASETS

Access of mass storage datasets can be restricted on two levels:

- Which users can access the dataset (privacy)
- What type of access is allowed (access mode)

The mass storage dataset protection system has two other dataset management aspects:

- Dataset use tracking
- Attribute association

Privacy

Mass storage permanent datasets fall into three categories, depending on which users can access the permanent dataset.

- *Private* datasets are accessible only to the dataset owner.

- *Semiprivate* datasets are accessible to the dataset owner and to a specific group of other users.

- *Public* datasets are accessible to all users.

New mass storage datasets are either public or private (not semiprivate) by default. Contact your Cray Research site analyst for the default value at your site. A new dataset can be explicitly declared as either public or private with the PAM (public access mode) parameter on the SAVE control statement. (See section 9.)

Access mode

In addition to establishing which users may access a dataset, the owner must establish what mode of access alternate users are allowed; that is, whether users other than the dataset owner may execute, read, write, or maintain the permanent dataset. Specifying the mode of alternate access depends upon what category of user is being granted the access. The three categories of users are:

- The dataset owner. The dataset owner is allowed all modes of access.

- Specific alternate users. Specific alternate users are named with the USER parameter of the PERMIT control statement (see section 9); the alternate user's allowed mode of access is declared with the AM (access mode) parameter of the same PERMIT control statement. Multiple PERMIT statements can be issued for the same permanent dataset to provide a list of alternate users. PERMIT can also be used to change or remove the allowed mode of access for an alternate user of the dataset. The allowed access mode for a specific user is known as a *permit*.

- All other users (the public). All users of a dataset not in the two categories above can be allowed (or denied) access to the dataset by using the PAM (public access mode) parameter on the ACQUIRE (section 10), SAVE, or MODIFY control statement (see section 9). The mode of public access to a dataset can be changed at any time with the MODIFY control statement.

Any mass storage permanent dataset can have a public access mode with any combination of permits. If an alternate user desiring access to a permanent dataset is allowed both public access and is named in a permit, the alternate user is allowed the access named in the permit. The permit takes precedence over the public access mode.

Such a combination of public and permitted access is often desirable. For example, suppose dataset FROG is to be used (executed as a program) by many groups of users, maintained by the dataset owner, and backed up or restored as needed by another user. Then, the dataset should have a public access mode of execute only and a permit of maintenance mode access for the alternate user who does dataset backup and restoration.

Note that all users, including the owner, must correctly specify any existing permission control words corresponding to the mode of access desired. For example, suppose dataset BIG has a public access mode of READ and a read password of README. Any user desiring to read the dataset must supply the read password (README) to gain access to the dataset. An exception occurs if the permanent dataset utilities are used. For more information, refer to section 11.

Dataset use tracking

The total access count and date/time of last access are recorded for each dataset in the Dataset Catalog (DSC). Access tracking capabilities include recording who accessed the dataset, how many times, and the date/time of last access. The permit mechanism described earlier in this section provides access tracking whenever a permit is issued for a user. A dataset that allows public access can also be tracked. However, the owner must explicitly state that public access tracking is required with the TA (track accesses) parameter on the ACQUIRE, SAVE, or MODIFY control statement; the system does not normally provide it.

Attribute association

The system allows permanent datasets having the same permanent dataset name (PDN) and additional identification (ID) to be distinguished by an edition number (ED). That is, there can be several datasets with different edition numbers that have the same PDN, ID, and ownership value.

A user permanent dataset is uniquely identified by the PDN, ID, ED and *ownership value*. The ownership value recorded in the DSC when a dataset is made permanent is normally equal to the user number as specified on the ACCOUNT or JOB control statement. Specific installations can choose to define dataset ownership as the account number rather than the user number. Contact your Cray Research site analyst to find out which type of ownership value is used.

Permanent mass storage datasets with the same PDN, ID, and ownership are assumed to be closely related. Therefore, most permanent dataset attributes are the same for all editions of the permanent dataset. The read, write, and maintenance permission control words, public access mode, public access tracking, and permits are the same for all datasets with the same PDN, ID, and ownership.

The text attribute is treated slightly differently. Any *text* supplied when the dataset is created is kept as a dataset attribute; if no *text* is supplied, the text attribute from the highest existing edition of the permanent dataset, if any, is used.

The notes attribute is treated similarly to text except that *notes* are assumed to be different for each dataset edition. *notes* supplied at dataset creation time are used; if no *notes* are supplied, none are used.

Deleting the data in a permanent dataset while leaving the dataset's name and attributes recorded in the Dataset Catalog (DSC) is possible. Such a dataset is referred to as a *partially deleted* dataset. Partial dataset deletion is described under Dataset Staging Control.

DATASET STAGING CONTROL

Staging is the process of transferring jobs and data in the form of COS datasets from a front-end computer system to Cray mass storage or of transferring datasets from Cray mass storage to a front-end computer system. Three control statements support staging datasets between COS and a front-end system: ACQUIRE, FETCH, and DISPOSE. Another control statement, SUBMIT, directs datasets to the COS input queue.

Verb	Function
ACQUIRE	Checks to see if the requested dataset is currently permanent on mass storage. If the dataset is already permanent, ACQUIRE works exactly like ACCESS (described earlier in this section) and allows dataset access to the job making the request. Alternatively, if the dataset is not mass storage resident, ACQUIRE obtains a front-end resident dataset, stages it to Cray mass storage, and makes it permanent and accessible to the job making the request. The dataset is staged from the front-end only if it is not already permanent.
DISPOSE	Directs a dataset to the specified queue for staging to a front-end system. DISPOSE can also be used to release a local dataset or to change dataset disposition characteristics.

Verb	Function
SUBMIT	Directs a dataset on Cray mass storage local to the submitting job to the COS input queue
FETCH	Obtains a front-end resident dataset and makes it local to the requesting job

The above control statements are fully described in section 10.

DISPOSE is invalid with tape datasets because DISPOSE applies only to the staging of datasets from mass storage to a front-end computer system.

Dataset control information such as save or access codes is usually required by a front-end system for management of its own files. Such control information can be sent by the Cray system user to the front-end system through the use of the text parameter (expressed as TEXT=*text*), which is a special parameter of the SAVE, MODIFY, ACQUIRE, FETCH, and DISPOSE statements. The content of the character string provided with the TEXT parameter is defined by the front-end system (see the appropriate station reference manual for the use of the TEXT parameter at your front-end system).

The *text* information not only provides most of the directives for obtaining the dataset from the front-end computer system but can contain sensitive or secure information as well. When using the ACQUIRE control statement, the staged dataset is recorded in the Dataset Catalog (DSC) and thus made permanent. Like any other mass storage permanent dataset, the staged dataset's attributes are recorded and protected as described under Protecting and Accessing Mass Storage Datasets, earlier in this section.

The owner of an acquired dataset can provide permission to acquire the dataset to other users by specifying a public access mode or by issuing permits. The actual dataset (that is, the data) need not reside on mass storage for the permissions to be issued. For this reason the *text*, as specified by the owner when the dataset was initially acquired, is retained by the system as an attribute. The owner can, at a later date, delete the data while still retaining all of the permanent dataset attributes. A dataset registered in the DSC in this manner is referred to as a *partially deleted* dataset.

When an authorized user acquires a partially deleted dataset, the text required to obtain the dataset from the front-end computer system is retrieved from the Dataset Catalog and sent along with the request. Therefore, the user need not specify the *text* in the ACQUIRE request. In fact, if the ACQUIRE is being issued by an alternate user as opposed to the owner, any *text* in the request is ignored. In this manner, the owner does not have to disclose the *text* information to other users.

The owner can at any time replace the *text* via the MODIFY command. After a partially deleted permanent dataset has been successfully acquired, the data is once again made permanent and is considered completely Cray mass storage resident. A subsequent ACQUIRE request, since the dataset is mass storage resident, is treated as an ACCESS request. Remember that the ACQUIRE request stages a dataset only if it is not already permanent on Cray mass storage.

PERMANENT DATASET UTILITIES

Three utilities (PDSDUMP, PDSLOAD, and AUDIT) can be used with any mass storage permanent datasets available to the user. Datasets processed by these utilities need not be local to the user job. The following utility routines are provided for mass storage permanent datasets.

Verb	Function
PDSDUMP	Dumps all specified permanent datasets to a user-specified dataset. Input and output datasets managed by the operating system can be included in the dump.
PDSLOAD	Loads permanent datasets that have been dumped by PDSDUMP and updates or regenerates the Dataset Catalog. Input and output datasets managed by the operating system can also be loaded with PDSLOAD.
AUDIT	Produces a report containing status information for each permanent dataset. AUDIT does not include system input or output datasets.

The above control statements are fully described in section 11.

LOCAL DATASET UTILITIES

Utility control statements provide the user with a convenient means of copying, positioning, or initializing local datasets. The following utilities are available to the user.

Verb	Function
COPYR, COPYF COPYD	Copies blocked records, files, and datasets, respectively
COPYU	Copies unblocked datasets

Verb	Function
SKIPR, SKIPF SKIPD	Skips blocked records, files, and datasets, respectively
REWIND	Positions a blocked or unblocked dataset at the beginning of data, that is, before the first word of the dataset
WRITEDS	Initializes a blocked random or sequential dataset. WRITEDS can also initialize a sequential dataset.

The above control statements are described in section 12.

ANALYTICAL AIDS

The following control statements provide analytical aids to the programmer.

Verb	Function
DUMPJOB DUMP	DUMPJOB and DUMP are generally used together to examine the contents of registers and memory as they were at a specific time during job processing. DUMPJOB captures the information so that DUMP can later format selected parts of it.
DEBUG	Produces a symbolic dump of the same data produced by DUMPJOB described above. DEBUG prints out the values of symbolic variables defined in the program being dumped.
DSDUMP	Dumps all or part of a blocked or unblocked dataset to another dataset in octal format
COMPARE	Compares two blocked datasets and lists all differences
FLODUMP	Dumps flowtrace tables when a program aborts with flowtrace active
PRINT	Writes the value of a JCL expression (as defined in section 16 of this manual) to the logfile
SYSREF	Generates a global cross-reference listing for one or more CAL or APML programs

Verb	Function
ITEMIZE	Inspects and generates statistics about library datasets. Libraries are described in section 5 of this manual; library dataset management is described under Object Library Management.

The above control statements are fully described in section 13.

EXECUTABLE PROGRAM CREATION

The LDR control statement calls the COS Relocatable Loader into execution. This utility prepares programs for execution from *relocatable modules*. A series of *relocatable modules* is normally created when a program is compiled or assembled. Each relocatable module normally represents one subroutine of the whole program, or the main program itself. Each relocatable module (also known as a *module*, an *object module*, a *relocatable*, or a *binary*) consists of a series of tables. The tables contain such information as executable machine (program) instructions, references to other modules (such as when one subroutine calls another), and the location of where the main program is to start execution.

Before a collection of relocatable modules (the program) can be executed, the collection of modules must be linked together into a single module. This single module, the *absolute load module*, contains the main program and a copy of every subroutine called, including ones found in the various system libraries. An absolute load module can be executed any time without having to be reprocessed by the Relocatable Loader. The COS Relocatable Loader executes as a utility program within the user field and provides the loading and linking in memory of relocatable modules from datasets on mass storage.

Very large programs might not fit in the available user memory space or might not use large portions of memory while other parts of the program are in execution. For such programs, the Relocatable Loader includes the ability to define and generate *overlays*--separate modules that the user creates and then calls and executes as necessary.

Executable program creation is fully described in section 14.

OBJECT LIBRARY MANAGEMENT

BUILD, a utility called through the BUILD control statement, creates and maintains object libraries.

Compiled subroutines (relocatable modules) can be collected into libraries that can be referred to later when creating a new program. COS provides several standard object libraries (see the Library Reference Manual, CRI publication SR-0014, for a description of the standard library routines available).

Any number of object libraries can be created, however, in addition to the ones supplied with COS.

Library datasets are designed primarily to provide the Relocatable Loader (see previous subsection) with a means of rapidly locating and accessing program modules. A *library dataset* is a dataset containing a program file followed by a directory file. The program file is composed of loader tables for one or more absolute or relocatable program modules. The directory file contains an entry for each program module.

BUILD is fully described in section 15.

JOB DEFINITION AND CONTROL 7

Several control statements allow the user to specify job processing requirements. This section contains the specifications for the following control statements used in defining a job and its operating characteristics to the operating system.

- JOB
- MODE
- EXIT
- MEMORY
- SWITCH
- *
- NORERUN
- RERUN
- IOAREA
- CALL
- RETURN
- ACCOUNT
- CHARGES
- ROLLJOB
- SET
- ECHO
- LIBRARY
- OPTION

JOB - JOB IDENTIFICATION

The JOB control statement defines the job to the operating system. It must be the first statement in a control statement file. The JOB control statement cannot be continued to subsequent lines or records. No leading blanks are allowed on the JOB statement. JOB is a system verb.

Format:

JOB,JN=*jn*,MFL=*fl*,T=*tl*,P=*p*,US=*us*,OLM=*olm*,CL=*jcn*,*gn*=*nr*.

Parameters are in keyword form; the only required parameter is JN.

JN=*jn* Job name. 1 through 7 alphanumeric characters. This name identifies the job and its subsequent output. JN is a required parameter.

MFL=*fl*† Maximum field length (decimal) allowed the job, in words. The job's maximum field length is set to the greater of *fl*, rounded up to the nearest multiple of 512 words, or the amount needed to load the Control Statement Processor (CSP). The job is aborted if the maximum field length is greater than the system maximum described below.

If this parameter is omitted, the maximum field length is set by the system to a value determined by an installation parameter.

If MFL is present without a value, the field length is the system maximum. The system maximum is the smaller of the total amount of memory available after the operating system is initialized minus the job's JTA size (see section 1) or an installation-defined maximum job field length.

T=*tl* Time limit (decimal) in seconds after which the job is terminated by the system. If this parameter is omitted, the time limit is set to a value determined by an installation parameter. If T is present without a value, a maximum of 16,777,215 seconds (approximately 194 days) is allowed.

P=*p* Priority level at which the job enters the system. This parameter can assume the values of 0 through 15 decimal. If P is 0, the job is not initiated. If omitted, a value specified by the installation is assumed.

US=*us* User number. 1 through 15 alphanumeric characters. The default is no user number. This parameter identifies the user submitting the job. Specific usage is installation defined.

OLM=*olm* Maximum size of $OUT. *olm* specifies a decimal count of 512-word blocks. A block holds about 45 print lines. The default and maximum values for *olm* are defined by the installation.

---

† The *fl* parameter on the JOB statement excludes the job's Job Table Area (JTA); space for the JTA is added by the system.

CL=*jcn* Name of the installation-defined job class where this job is to be placed. 1 through 7 alphanumeric characters. The job is aborted if it does not fit the requirements of the indicated class or if the indicated class does not exist. The default is no class name.

gn=*nr* Type and number of dedicated resources required by a job.

gn is a generic resource name of 1 through 7 alphanumeric characters. A generic resource name corresponds to a device type. For example, a generic name of SSD could be given to a Solid-state Storage Device. Generic names are defined by site administration. COS provides one generic name (*TAPE, which refers to a dual density tape unit capable of 1600 or 6250 bpi), but sites may define up to 16 generic names. Contact your Cray Research site analyst for the generic names used at your site.

nr is a positive integer and represents the maximum amount of the associated resource that may be used concurrently during job execution; the default is 0. A job is initiated only when the amount of each resource reserved is eligible for use. The job is aborted if it attempts to access more resources than are reserved with the JOB control statement.

nr is the decimal number of units of the specified resource type. If *gn* refers to a tape device type, *nr* is the number of tape units to be used concurrently. If *gn* refers to a disk device type, *nr* is the decimal number of sectors required.

MODE - SET OPERATING MODE

The MODE control statement allows the user to set or clear mode flags in the Exchange Package for the job. MODE is a system verb.

Format:

```
MODE,FI=option,BT=option,EMA=option,AVL=option,ORI=option.
```

Parameters are in keyword form. At least one parameter must be specified. The parameters are:

FI=*option* Floating-point interrupt mode. *Option* can be either:

ENABLE	Enable floating-point error interrupts; default.
DISABLE	Disable floating-point error interrupts; floating-point errors are ignored.

BT=*option* Bidirectional transfer mode. The BT parameter is used on CRAY X-MP Series Computer Systems only. *option* can be either:

ENABLE	Enable bidirectional memory transfers; default.
DISABLE	Disable bidirectional memory transfers; block reads and writes are not performed concurrently.

EMA=*option*

Extended memory addressing mode. The EMA parameter is used on CRAY X-MP† Series Computer Systems only; it causes an abort on CRAY-1 dystems. *option* can be either:

ENABLE	Enables extended memory addressing.
DISABLE	Disables extended memory addressing; default is an installation option, released as EMA=DISABLE. On the CRAY X-MP model 48, the default is released as EMA=ENABLE.

AVL=*option*

Second vector logical functional unit mode. The AVL parameter is used on CRAY X-MP† Series Computer Systems only; it causes an abort on CRAY-1 systems. *option* can be either:

ENABLE	Makes available two logical functional units, the first of which shares reservation logic with the vector floating multiply unit.
DISABLE	Makes available only one vector logical unit. The vector multiply reservation path is not shared; default is an installation parameter, released as AVL=DISABLE.

ORI=*option*

Operand range error interrupt mode. The ORI parameter is used on CRAY X-MP Series Computer Systems only; *option* can be either:

ENABLE	Enables interrupts on operand range errors; default.
DISABLE	Disables interrupts on operand range errors.

† Not available on all CRAY X-MP systems. Check with a site analyst to determine if this feature is available.

EXIT - EXIT PROCESSING

An EXIT control statement indicates the point in the control statement file where processing of control statements resumes following a job step abort from a program. If no job step abort occurs, the EXIT control statement indicates the end of the control statement processing. EXIT is a system verb.

Format:

```
EXIT.
```

Parameters: None

MEMORY - REQUEST MEMORY CHANGE

The MEMORY control statement allows the user to request a new field length and/or mode of field length reduction. Job memory management is further discussed in section 3.

MEMORY is a system verb.

Format:

$$\text{MEMORY}[,\text{FL}=fl]\left[\left\{\begin{matrix},\text{USER}\\ \text{AUTO}\end{matrix}\right\}\right].$$

The keywords USER and AUTO are mutually exclusive. However, at least one of the following three parameters must be specified:

FL=*fl*	Field length. *fl* specifies the decimal number of words of field length to be allocated to the job. If FL is specified without a value, the new field length is set to the maximum allowed the job.
USER	Field length reduction is managed by the user (user mode)
AUTO	Field length reduction is managed by the system (automatic mode)

The job's field length can be changed by using the FL parameter. The field length is set to the larger of the requested amount rounded up to the nearest multiple of 512 words or the smallest multiple of 512 decimal words large enough to contain the user code/data, LFT, DSP and buffer areas. Field length management is in user mode for the duration of the next job step.

The management of a job's field length can be changed by using either the USER or AUTO parameters. When the USER parameter is specified, the job is placed in user mode until a subsequent request is made to return it to automatic mode. When the AUTO parameter is specified, the job is placed in automatic mode.

The job step is aborted if completing the request results in a field length greater than the maximum allowed the job. The maximum is the smaller of the total number of words available to user jobs minus the job's JTA or the amount determined by the MFL parameter on the JOB statement.

Examples:

```
MEMORY,FL,USER.
```

The job's field length is set to the maximum allowed the job and the job is placed in user mode until an explicit request is made to return it to automatic mode.

```
MEMORY,AUTO.
```

The job is returned to automatic mode. Its field length is reduced at the next job step.

```
MEMORY,FL=28988.
```

The field length is adjusted. If the job is in user mode by explicit user request, no change in mode occurs; otherwise, the job is placed in user mode for the duration of the next job step.

```
MEMORY,FL=28988,AUTO.
```

The field length is adjusted and the job is placed in user mode for the duration of the next job step. After the next job step, the job is put in automatic mode.

SWITCH - SET OR CLEAR SENSE SWITCH

The SWITCH control statement allows a user to turn on or turn off pseudo sense switches. SWITCH is a system verb.

Format:

```
SWITCH,n=x.
```

Parameters:

n	Number of switch (1 through 6) to be set or cleared	
x	Switch position	
	ON	Switch *n* is turned on; set to 1.
	OFF	Switch *n* is turned off; set to 0.

* - COMMENT STATEMENT

The comment control statement allows the user to annotate job control statements with comments. A terminator is not required on a comment control statement. * is a system verb.

Format:

```
* comment text
```

Parameters: None

NORERUN - CONTROL DETECTION OF NONRERUNNABLE FUNCTIONS

The NORERUN control statement allows the user to specify whether the operating system is to recognize functions that would make a job rerunnable. The current rerunnability of the job is not affected. NORERUN is a system verb.

Format:

$$\boxed{\text{NORERUN,}\begin{Bmatrix}\text{ENABLE}\\ \text{DISABLE}\end{Bmatrix}.}$$

The keywords ENABLE and DISABLE are mutually exclusive. The default for the system as released is NORERUN,ENABLE; however, this is an installation option.

Selecting ENABLE instructs the system to begin monitoring functions performed by the job and to declare the job nonrerunnable if any of the nonrerunnable functions are performed.

Selecting DISABLE instructs the system to stop monitoring functions for nonrerunnable operations. If a job has already been declared to be nonrerunnable, specifying DISABLE does not make the job rerunnable again.

RERUN - UNCONDITIONALLY SET JOB RERUNNABILITY

The RERUN control statement allows the user to unconditionally declare a job to be either rerunnable or nonrerunnable. If RERUN is used to declare a job rerunnable, the subsequent execution of a nonrerunnable function may cause the system to declare the job nonrerunnable, depending on whether a NORERUN control statement or macro is also present. RERUN is a system verb.

Format:

$$\boxed{\text{RERUN,}\begin{Bmatrix}\text{ENABLE}\\ \text{DISABLE}\end{Bmatrix}.}$$

The keywords ENABLE and DISABLE are mutually exclusive. If no parameter is specified on the control statement, installation option determines if the job is to be rerunnable; the default for the system as released is RERUN,ENABLE.

If ENABLE is selected, the system is instructed to consider the job to be rerunnable, regardless of what functions have been executed previously.

If DISABLE is selected, the system marks the job not rerunnable regardless of what functions have been executed previously.

The RERUN control statement does not affect the monitoring of the user job for nonrerunnable functions.

IOAREA - CONTROL USER'S ACCESS TO I/O AREA

The IOAREA control statement locks (denies the user access to) or unlocks (gives the user access to) that portion of the user field containing the user's Dataset Parameter Area (DSP) and I/O buffers. This area follows the High Limit Memory address (HLM) of the user field. The user of the stack version of the COS libraries needs to note that IOAREA does not protect I/O buffers or DSPs that have been allocated within the user's stack space. IOAREA is a system verb.

Format:

$$\text{IOAREA,}\begin{Bmatrix}\text{LOCK}\\ \text{UNLOCK}\end{Bmatrix}.$$

The keywords LOCK and UNLOCK are mutually exclusive. A parameter must be specified on the control statement. When the control statement is not used, the user's I/O area is assumed to be unlocked.

If LOCK is selected, the system sets the limit address to the base of the DSPs, thereby denying direct access to the user's DSP area and I/O buffers. When the I/O area is locked, the library I/O routines make a system request to gain access to the I/O area. Although the system request introduces additional overhead in job processing, it should prevent accidental destruction of the I/O area.

If UNLOCK is selected, the system sets the limit address to the value specified in JCFL, allowing access to the user's DSP area and I/O buffers.

CALL - READ CONTROL STATEMENTS FROM ALTERNATE DATASET

The CALL control statement instructs COS to begin reading control statements from the first file of the indicated dataset. CALL can appear anywhere in the control statement file. Nesting of CALL statements to seven levels is allowed. COS reads and processes the control statements from the indicated dataset until COS encounters an end-of-file or a RETURN statement. Control then reverts to the previous control statement dataset.

CALL does not rewind the specified dataset before reading it.

The indicated dataset can contain either simple control statements or a procedure definition. Simple control statements are executed without any parameter substitution. On the other hand, parameter substitution is possible when the indicated dataset contains a procedure definition. The optional CNS parameter on the CALL statement allows COS to determine the form of control statements used. If CNS is not present, the statements on the indicated dataset are assumed to be simple control statements. In this case, they are executed exactly as read from the dataset, beginning with the first statement.

However, if CNS is present on the CALL statement, the statements on the dataset are treated as a procedure definition. This means that parameter substitution can be performed before executing the statements. In this case, the first statement is assumed to be a prototype statement and subsequent statements are the procedure body definition. If the indicated dataset contains a procedure definition, the dataset will be closed after parameter substitution and before invocation of the procedure.

If the dataset contains a procedure definition, the PROC and ENDPROC statements must not enclose the definition, unlike a procedure defined in-line within a control statement file. The PROC and ENDPROC statements may appear within the definition. Any statement enclosed by PROC and ENDPROC becomes a procedure definition which is included in the $PROC system procedure dataset when the enclosing procedure is invoked by a CALL statement. The enclosing procedure is not added to the $PROC dataset.

When the CNS option is used and the procedure definition contains a nested PROC/ENDPROC sequence, the parameter substitution performed according to the prototype statement for the outermost procedure definition (the first statement of the dataset) is also performed on all nested definitions. This can produce warning messages if the inner definitions use keywords or positional parameters different from those specified for the outer definition. The nested definitions are written to $PROC with all matching substitutions performed and all nonmatching substitutions retained in the original form.

CALL is a system verb.

Format:

CALL,DN=*dn*[,CNS].

Parameters are in keyword form.

DN=*dn* Name of dataset from which to begin reading control statements. This is a required parameter.

CNS Crack next statement. This is an optional parameter. If present, the first statement on the dataset named by DN will be treated as the prototype statement for the procedure whose body is defined by the remaining statements in the first file of the dataset, and the next statement in the control statement dataset containing the CALL statement will be read by COS and treated as an invocation of the procedure. Parameters supplied on that statement are substituted according to the rules of parameter substitution described in section 16 of this manual.

Examples:

1. Use of CALL without CNS:

Assume that dataset X contains the following control statements:

```
ACCESS,DN=A,PDN=B,UQ.
DELETE,DN=A.
RELEASE,DN=A.
```

If dataset B has been previously saved, then the result of the statement:

```
CALL,DN=X.
```

would be:

```
ACCESS,DN=A,PDN=B,UQ.
PD000 - PDN = B                ID =      ED =  1   OWN = ABC
PD001 - ACCESS COMPLETE
DELETE,DN=A.
PD000 - PDN = B                ID =      ED =  1   OWN = ABC
PD001 - DELETE COMPLETE
RELEASE,DN=A.
```

2. Use of CALL with CNS:

Assuming the contents for dataset X are the same as in example 1, the result of the statement:

```
CALL,DN=X,CNS.
```

would be:

```
ACCESS,DN=A,PDN=B,UQ.
CS109 - POSITIONAL PARAM. AFTER KEYWORDS IN PROTOTYPE: UQ
*,DN=A.
CS122 - NO VALUE WAS ASSIGNED TO UQ
AB025 - USER PROGRAM REQUESTED ABORT
AB000 - JOB STEP ABORTED.     P = 00000743b
```

In this case the CNS parameter causes COS to consider the ACCESS statement to be a prototype statement; the DN, PDN and UQ keywords are assumed to be the identifiers of substitutable parameters.

3. Valid CALL with CNS without nested definitions:

   Assuming that the contents of dataset X are:

```
D,A,B.
ACCESS,DN=&A,PDN=&B,UQ.
DELETE,DN=&A.
RELEASE,DN=&A.
```

   If the permanent dataset EXAMPLE exists, the result of the statements:

```
CALL,DN=X,CNS.
*,DS,EXAMPLE.
```

   would be:

```
ACCESS,DN=DS,PDN=EXAMPLE,UQ.
PD000 - PDN = EXAMPLE          ID =       ED =  1   OWN = ABC
PD001 - ACCESS COMPLETE
DELETE,DN=DS.
PD000 - PDN = EXAMPLE          ID =       ED =  1   OWN = ABC
PD001 - DELETE COMPLETE
RELEASE,DN=DS.
```

4. CALL with a nested PROC/ENDPROC definition:

   Assuming that dataset X contains the following statements:

```
D,A,B.
PROC.
A,Q,B.
ACCESS,DN=&Q,ID=&B.
ENDPROC.
ACCESS,DN=&A,ID=&B,UQ.
DELETE,DN=&A.
RELEASE,DN=&A.
```

If permanent dataset Z with ID D exists, the result of the statements:

```
CALL,DN=X,CNS.
*,Z,D.
```

would be:

```
CS125 - NO SUCH FORMAL PARAMETER: Q
<DEFINITION>    PROC.
<DEFINITION>    A,Q,B.
<DEFINITION>    ACCESS,DN=&Q,ID=D.
<DEFINITION>    ENDPROC.
ACCESS,DN=Z,ID=D,UQ.
PD000 - PDN = Z                 ID =      ED =  1   OWN = ABC
PD001 ACCESS COMPLETE
DELETE,DN=Z.
PD000 - PDN = Z                 ID =      ED =  1   OWN = ABC
PD001 - DELETE COMPLETE
RELEASE,DN=Z.
```

The $PROC dataset would contain a procedure with the definition:

```
A,Q,B.
ACCESS,DN=&Q,ID=D.
```

The &B in the original definition was replaced by the value which was specified for the corresponding parameter B in the outermost procedure. The &Q was retained, since there was no corresponding replacement in the outermost procedure.

RETURN - RETURN CONTROL TO CALLER

The RETURN control statement returns control to the caller. The caller can be a procedure or the job's control statement file. Processing resumes with the caller's next control statement. A RETURN control statement can be embedded anywhere within the called procedure. However, a RETURN control statement need not be placed at the end of the procedure because an end-of-file record is interpreted as the control statement sequence of an EXIT, RETURN, and RETURN,ABORT. A RETURN encountered in the primary control statement file is ignored. RETURN is a system verb.

Format:

```
RETURN[,ABORT].
```

Parameter:

ABORT — After returning to the previous control statement level, ABORT causes COS to issue a job step abort. ABORT is an optional parameter.

ACCOUNT - VALIDATE USER NUMBER AND ACCOUNT

The ACCOUNT control statement validates the job's user number, user password, account number, and account password. A job is processed only if the user number/password pair and the account number/password pair (if specified) are valid. The ACCOUNT control statement provides accounting data for the installation. Moreover, the installation and the individual users are also provided means for ensuring both privacy and security through the use of ACCOUNT parameters.

The ACCOUNT statement declares the user's account and charge numbers to COS. It must immediately follow the JOB control statement if the installation has defined accounting or security as mandatory. Only one ACCOUNT statement is allowed per job. ACCOUNT is a system verb.

If the job is interactive, and accounting is mandatory, the ACCOUNT statement must be the first statement entered in a session. If it is not, a prompt is issued to the terminal requesting the ACCOUNT statement. A similar prompt is issued for syntax errors made on the ACCOUNT statement.

NOTE

The ACCOUNT control statement parameters do not appear with the ACCOUNT control statement in the job logfile.

Format:

ACCOUNT,AC=*ac*,APW=*apw*,NAPW=*napw*,US=*us*,UPW=*upw*,NUPW=*nupw*.

Parameters are in keyword form. The only required parameter is AC; the installation defines whether one or more passwords are needed.

The installation generally sets up AC, APW, US, and UPW parameters. However, the user specifies NAPW and NUPW. Including a new account password provides the user accounting protection, since only the person who knows the NAPW can run a job under a given user's account number. NUPW is an additional security check available to the user. Therefore, NAPW and NUPW values should be known only to the individual user who specifies them.

AC=*ac*	Account number. 1 through 15 alphanumeric characters assigned to the user. This number identifies the user for accounting purposes, and is a required parameter. The account number is not the same as the user number on the JOB control statement, unless the site chooses to use the same characters for both numbers.
APW=*apw*	Account password. 1 through 15 alphanumeric characters or null. A password must be specified if the installation has made the password mandatory by installation parameter. To change a null account password to a non-null account password, a user must specify the keyword APW without a value and NAPW with the new value.
NAPW=*napw*	New account password. 1 through 15 alphanumeric characters or null. This new password replaces the old account password if the account number/password pair given by the AC and APW parameters is valid. NAPW may be specified without a value to change the account password to null. To change a null account password to a non-null account password, APW must be specified without a value and NAPW must be specified with the new value.
US=*us*	User number. 1 through 15 alphanumeric characters assigned to the user. This number identifies the user for system access purposes and is an optional parameter. The user number is not the same as the account number, unless the site chooses to use the same characters for both numbers. This parameter, if specified, overrides the user number on the JOB control statement. If US is not specified on the ACCOUNT control statement, the user number on the JOB statement is used by COS.
UPW=*upw*	User password. 1 through 15 alphanumeric characters. A password must be specified if the installation has made security checking mandatory.
NUPW=*nupw*	New user password. 1 through 15 alphanumeric characters. This new password replaces the old user password *upw* if the user number/password pair given by the US and UPW parameters is valid.

CHARGES - JOB STEP ACCOUNTING

The CHARGES control statement allows the user to monitor a job's usage of computer resources up to a specific point in a job. Hence, CHARGES can be used for either partial or total resource reporting.

Partial reporting occurs when parameters are specified on the control statement. In this case, usage statistics for the computer resources specified on the CHARGES statement are obtained for the job steps preceding the CHARGES statement. The summary is placed in the user log and the system log.

Total reporting occurs when usage statistics are obtained for all the resources in all the available resource groups. The summary is placed in the user log and the system log.

CHARGES is automatically invoked when a job terminates so that usage statistics of the entire job are reported.

Format:

```
CHARGES,SR=options.
```

Parameters are in keyword form.

SR=*options*

System resources used. Any one or more of the following groups of resources can be specified. Options are separated by colons. The default is a listing of the job's usage of resources in all of the following groups:

- JNU Job name and user number
- DS Permanent dataset space accessed, permanent dataset space saved, temporary dataset space used, 512-word disk blocks (sectors) moved, user I/O requests, memory-resident datasets used, number of OPEN calls and number of CLOSE calls
- WT Time waiting in the input queue before beginning execution

MM Minimum job size (words), maximum job size (words), execution-time memory usage in million word-seconds, I/O wait-time memory usage in million word-seconds, maximum field length used (words), minimum field length used (words), maximum JTA used (words), and minimum JTA used (words)

CPU Time executing in CPU, I/O waiting time, and time waiting for CPU. CPU gives the totals for the entire job.

NBF Number of 512-word blocks (sectors) received from a front end and number of 512-word blocks (sectors) queued to a front end

TASK Time executing in CPU, I/O wait time, and time waiting for CPU. The TASK option breaks down the time information according to user task number, and provides a total for the entire job.

TPS Number of tape devices reserved, number of tape volumes mounted, amount of tape data moved (expressed as a multiple of 512 words) and number of physical tape blocks moved

FSU† Fast storage usage. Amount of SSD or BMR (Solid-state Storage Device or Buffer Memory) space reserved and used.

The amounts are returned as two values; one is the wall-clock time times the reserved space usage amount and the other is CPU time multiplied by the reserved space usage amount for each device. Any of the four usage amounts, if nonzero, are placed in the user logfile.

ROLLJOB - ROLL A USER JOB TO DISK

The ROLLJOB control statement allows the user to protect a job by writing it to disk so that it can be recovered in case a system interruption occurs. ROLLJOB is a system verb.

† Deferred implementation

Format:

```
ROLLJOB.
```

Parameters: None

SET - CHANGE SYMBOL VALUE

The SET control statement changes the value of a specified valid job control language symbol. Valid symbols are those classified as alterable by the user (U) in table 16-1. A job step abort occurs if a symbol included in a SET control statement is unknown to the system, can be set only by COS, or is a constant. SET is a system verb.

Format:

SET(*symbol*=*expression*)

Parameters:

symbol A valid user-alterable symbol; *symbol* is a required parameter.

expression
A valid arithmetic, logical, or literal assignment expression. It may be delimited with parentheses to simplify interpretation during control statement evaluation. *expression* is a required parameter.

Examples:

```
SET(J1=J1+1)
```

This example increases the procedure-local register J1 by 1.

```
SET(G1=(SYSID.AND.177777B))
```

The global register G1 is given an ASCII value that is the low-order two characters from the current system revision level (COS *X.XX*).

SET(G3=((ABTCODE.EQ.74).AND.(G2.EQ.0)))

The global register G3 is assigned a value, depending upon the current values of ABTCODE and G2.

ECHO - ENABLE OR SUPPRESS LOGFILE MESSAGES

The ECHO control statement allows the user to control the message classes to be written to the user's logfile by turning the classes ON or OFF. ECHO may be used more than once during a job to toggle the printing or suppression of message classes. ECHO is a system verb.

Format:

ECHO,ON=$class_1$:...:$class_n$,OFF=$class_1$:...:$class_n$.

Parameters are in keyword form.

ON=$class_i$ When a program or the operating system issues messages, they are written to the user's logfile in the classes specified. If any other classes were specified but not turned off by this statement, the union of the two sets of classes is enabled. If the ECHO control statement contains only the keyword ON or ON=ALL, all messages are written to the logfile. This is the default for the start of a job.

OFF=$class_i$
Messages in the classes specified are not written to the user's logfile. If any other classes were specified but not turned on by this statement, the union of the two sets of classes is suppressed. If the ECHO control statement contains only the keyword OFF or OFF=ALL, all messages in the defined classes are supressed.

Messages that are not classified may not be turned off.

The only classes that the operating system acknowledges are the following:

Class	Description
JCL	Messages that originate in the user's JCL input file
ABORT	AB*xxx* and system traceback messages that COS issues when a job fails
PDMINF	Dataset information messages produced by PDM
PDMERR	Error messages produced by PDM

The keywords ON and OFF may be used in any combination. However, ensure that the classes specified do not overlap between the keywords, and that both defaults are not included.

When a job calls a procedure, the echo state of the job is the same upon return from the procedure as before, even though the procedure may use a different echo state. The following occurs when ECHO is used with CALL and PROC:

- The echo state of the caller (a job or another procedure) is saved so that on return to the caller the same state is in effect as before the call.
- When the procedure includes an ECHO statement, the new echo state is in effect only for the duration of the procedure. If the procedure does not include an ECHO statement, the echo state of the caller is in effect.

LIBRARY - LIST AND/OR CHANGE LIBRARY SEARCHLIST

The LIBRARY control statement allows the user to specify the library datasets to be searched during the processing of control statement verbs. LIBRARY also allows the user to list the current or new searchlist to the logfile for verification.

When modifying the searchlist, the current members of the searchlist can be retained in the new searchlist by including an asterisk in the LIBRARY control statement. The asterisk corresponds to all members of the current searchlist in their present order. If the asterisk is omitted, the new searchlist contains only the library dataset names identified on the LIBRARY control statement. LIBRARY is a system verb.

The default library searchlist upon job initiation consists of the single library dataset $PROC.

Format:

```
LIBRARY,DN=dn1:dn2...:dn64,V.
```

Parameters:

DN=dn_i Library dataset names to become members of the new library searchlist. A maximum of 64 names (separated by colons) can be specified. The order in which they appear is the order they are searched. An asterisk included in the list signifies the current searchlist members are to be part of the new searchlist in their current order.

V List the current library searchlist on the logfile for verification. When specified along with the new searchlist, the new searchlist is listed.

OPTION - SET USER-DEFINED OPTIONS

The OPTION control statement allows the user to specify user-defined options, such as the format of the job's listing. OPTION is a system verb.

Format:

$$\text{OPTION,[LPP=}n\text{,]}\left[\text{STAT=}\begin{Bmatrix}\text{ON}\\ \text{OFF}\end{Bmatrix}\right]\text{.}$$

Parameters:

LPP=n Number of lines per page; a decimal number from 0 through 255. If 0 is specified, the current number of lines per page is not changed. The default is an installation parameter.

STAT=$\begin{Bmatrix}\text{ON}\\ \text{OFF}\end{Bmatrix}$ STAT=ON has two effects. First, it enables accounting for any mass storage datasets created while STAT=ON is in effect; statistics are reported separately for each device containing all or part of such datasets. Second, it enables the printing of the dataset I/O statistics collected for all datasets to user $LOG at release time.

STAT generates a short- or long-form I/O statistic, depending on when the option is invoked. The short-form consists of the dataset name, device name, dataset size, number of user I/O requests, number of 512-word blocks transferred, and total time blocked for I/O for the dataset.

```
SY005 - $IN  DD-A2-24  512W  1R  1S  .248SEC
```

The long-form output line consists of the dataset name, device name, size in sectors, number of read requests, number of sectors read, number of write requests, number of sectors written, and time.

```
SY005 - TEST  DD-A1-32  1S  0RR  0SR  1WR  1SW  .05S
```

No statistics are collected or printed if STAT=OFF, which is the default condition. However, datasets created while STAT=OFF, then released while STAT=ON have the short-form output line printed out.

Examples:

1. ```
   ASSIGN,DN=X.
   OPTION,STAT=ON.
   COPYF,.....,O=X.
   RELEASE,DN=X.
   ```

   Short-form I/O statistics are printed for X.

2. ```
   OPTION,STAT=ON.
   ASSIGN,DN=X.
   COPYF,.....,O=X.
   RELEASE,DN=X.
   ```

   Long-form I/O statistics are printed for X.

3. ```
   OPTION,STAT=ON.
   ASSIGN,DN=X.
   COPYF,......,O=X.
   OPTION,STAT=OFF.
   RELEASE,DN=X.
   ```

   No I/O statistics are printed for X, even though statistics were collected.
   ```

4.
```
OPTION,STAT=ON.
ASSIGN,DN=X.
COPYF,....,O=X.
OPTION,STAT=OFF.
...
OPTION,STAT=ON.
RELEASE,DN=X.
```

Long-form I/O statistics are printed for X.

DATASET DEFINITION AND CONTROL 8

Datasets are defined and managed by the user through three dataset control statements: ASSIGN, ACCESS, and RELEASE. ACCESS is not used for Integrated Support Processor (ISP) datasets. Instead, the ISP control statement gives a user job access to an ISP, and the CONNECT control statement accesses a specific dataset. Refer to the ISP General Information Manual, CRI publication SG-0094, for details†.

- ASSIGN defines characteristics for datasets. ASSIGN also can be used to create a mass storage dataset.
- ACCESS (described in section 9) makes an existing disk or tape permanent dataset local to a job or can be used to create a dataset on magnetic tape.
- RELEASE relinquishes access to the named dataset for the job.

ASSIGN - ASSIGN DATASET CHARACTERISTICS

The ASSIGN control statement creates a mass storage dataset and assigns dataset characteristics for tape and mass storage. If an ASSIGN is used for dataset creation, it must appear before the first reference to the dataset; otherwise, the characteristics are defined at the first reference. If an ASSIGN is used for a tape dataset, it must follow the tape ACCESS request. ASSIGN†† is a system verb.

† Publication SG-0094 will be available when the ISP software is released.

†† ASSIGN does not create a dataset that the CFT OPEN statement recognizes as existing unless the FILE parameter is on the OPEN statement. See CRI publication SR-0009, FORTRAN (CFT) Reference Manual.

Format:

```
ASSIGN,DN=dn,S=size,SZ=size,NOF,BS=bsz:bpt,DV=ldv,DT=dt,DF=df,
RDM,U,MR,LM=lm,INC=nds,C,DC=dc,BFI=bfi,A=FTxx,FD=fd,CV=cv,
CS=cs,F=f,RF=rf,RS=rs,MBS=mbs.
```

Parameters are in keyword form. The only required parameter is DN.

DN=*dn* Local dataset name. 1 through 7 alphanumeric characters, the first of which is A through Z, $, %, or @; remaining characters may also be numeric. DN is a required parameter.

S=*size* Dataset size. Octal number of sectors (512-word blocks) to be reserved for the dataset. If the dataset size is not given, the space for the dataset is dynamically allocated as needed. S and the SZ option are mutually exclusive. Furthermore, S applies to mass storage datasets only, and is ignored when used for magnetic tape datasets.

SZ=*size* Dataset size. Decimal number of sectors (512-word blocks) to be reserved for the dataset. If the DV option specifies a generic resource or if *ldv* is a controlled device, SZ is the largest number of sectors associated with this dataset which can reside on the device. The mass storage space reservation occurs when the ASSIGN command is processed. If the SZ option is not specified, the space for the dataset is dynamically allocated as needed. S and the SZ option are mutually exclusive. SZ applies to mass storage datasets only and is ignored when used for magnetic tape datasets.

Although the SZ option is specified as decimal sectors, disk space is allocated by COS in tracks which are larger than sectors. When an ASSIGN statement declares dataset size, COS rounds the sector count up to an integral multiple of track size and allocates that number of tracks. For example, when ASSIGN(...,S=1,...) is specified, COS allocates one track to the dataset, even though the request is for one sector. If the dataset resides on a DD-19 or a DD-29, a track is equivalent to 18 decimal sectors. Other disk devices can have different track sizes.

When the disk device specified on the ASSIGN statement is a controlled device with a generic name, the total concurrent use of the device must be declared on the JOB statement as decimal sectors. If the space on the device is divided among several datasets with the SZ option on the ASSIGN statement, a rounding error may occur with each use of the SZ or S options. The result can be an unexpected GENERIC RESOURCE LIMIT EXCEEDED error or an unexpected device overflow. The SZ option can produce other results when it is used with the NOF parameter of ASSIGN. Those results are described under NOF in this section.

If both INC and SZ are specified, SZ is used initially and INC is used subsequently.

To divide space among several datasets on a generic resource such as Buffer Memory or Solid-state Disk, sector counts should be specified as multiples of track size. Track size is currently 18 decimal for all controlled devices.

NOF — No overflow. When NOF is indicated, the dataset does not span any more than the specified device. The SZ and NOF options on the ASSIGN statement produce the following:

SZ and NOF specified: abort at MIN (Remaining Job Limit, SZ)

SZ specified without NOF: overflow at MIN (Remaining Job Limit, SZ)

NOF specified without SZ: abort at Remaining Job Limit

Neither SZ nor NOF specified: overflow at Remaining Job Limit

BS=*bsz* — Buffer size and partitioning value. The value given to *bsz* specifies the size of a dataset's circular I/O buffer in 512-word blocks. The default is the value defined by the installation parameter. The U and BS parameters are mutually exclusive.

BS=*bsz:bpt*†

The value given *bpt* specifies the minimum size in 512-word blocks for transfers to and from the circular buffer. This value must be less than or equal to the buffer size. The default transfer size is half a buffer.

DV=*ldv* Logical device on which the dataset begins. If a logical device name is not given, one is chosen by the system. *ldv* can also be a generic resource name. Consult site operations for possible logical device names and generic resource names. This parameter applies to mass storage datasets only and is ignored when used for magnetic tape datasets.

DT=*dt* Device type. The allowable device types are CRT (interactive) and MS (mass storage). MS is the default. This parameter is ignored when used for magnetic tape datasets.

DF=*df* Dataset format. This parameter is used only on output; it is valid only when DT=CRT. This parameter is ignored when used for magnetic tape datasets. Two formats are supported:

- CB Character blocked. End-of-record RCWs are converted by the station to the format which the station supports. CB is the default.

- TR Transparent. End-of-record RCWs are not converted. The user is responsible for inserting cursor controls.

RDM Random dataset. If the RDM parameter is present, the dataset is read and written randomly (that is, records may be read or written out of sequence). If the RDM parameter is not specified, only sequential or FORTRAN direct access I/O is allowed on the datasets. This parameter applies to mass storage datasets only and is invalid for magnetic tape datasets.

U Unblocked dataset structure. If the U parameter is present, the dataset is not in COS-defined blocked format. If the U parameter is absent, the dataset is a COS blocked dataset. (See section 2 for information on unblocked dataset format.) This parameter is invalid for interchange format tape datasets. The U and BS parameters are mutually exclusive.

† Deferred implementation

MR Memory-resident dataset. If this parameter is present, the system I/O routines write the buffers to mass storage only if they become full. If the MR parameter is absent, the dataset is not a memory-resident dataset. MR generates an error if the U parameter is specified. This parameter applies to mass storage datasets only and is invalid for magnetic tape datasets.

LM=*lm* Maximum size limit for this dataset. *lm* specifies a decimal count of 512-word blocks. The job step will be aborted if this size is exceeded. The default and maximum dataset size limits are set by an installation parameter. This parameter applies to mass storage datasets only and is ignored for magnetic tape datasets.

INC=*nds* Number of decimal sectors to allocate each time allocation occurs. If both INC and SZ are specified, SZ is used initially and INC is used subsequently.

C Contiguous space allocation. Use C to allocate contiguous space requested by the SZ or INC parameter or the default size. If C is not specified, the system tries to find contiguous space on the selected device only. If C is specified, the system searches on every eligible device.

If contiguous space cannot be found when C has been specified, the return status SPACE NOT AVAILABLE appears.

DC=*dc* Disposition code. Disposition to be made of the dataset when it is released. This parameter applies to mass storage datasets only and is ignored for tape datasets. The default is SC.

dc is a 2-character alphabetic code describing the destination of the dataset as follows:

IN The dataset is placed in the input queue of the destination station.

ST Stage to mainframe. Dataset is made permanent at the mainframe of job origin.

SC Scratch dataset. Dataset is deleted.

PR Print dataset. Dataset is printed on printer at the mainframe of job origin.

PU Punch dataset. Dataset is punched on any card punch available at the mainframe of job origin.

PT Plot dataset. Dataset is plotted on any available plotter at the mainframe of job origin.

MT Magnetic tape. Dataset is written on magnetic tape at the mainframe of job origin.

BFI=*bfi* Blank field initiation. Octal representation of ASCII code indicating the beginning of a sequence of blanks. BFI=OFF means that blank compression is inhibited. The default code is 33_8 (ASCII ESC code) but can be changed by an installation parameter. BFI is ignored for ISP datasets.

A=FT*xx* Unit name. Unit names allow the user to refer to a dataset from a FORTRAN program. Each unit name is 4 characters in the form FT*xx*, where *xx* is the unit number specified.

The unit number is an integer value in the range 0 through 102. However, because unit numbers 100, 101, and 102 are reserved for system use, a user may designate unit numbers 0 through 99.

Use of this parameter associates the designated unit with the dataset specified by the DN parameter. At job initiation, unit FT05 is associated with dataset $IN and unit FT06 is associated with dataset $OUT. Unit names should not be used as dataset names.

NOTE

If a dataset name is used in place of a unit name or vice versa, FORTRAN '77 auxiliary statements (that is, OPEN, CLOSE, and INQUIRE) produce unpredictable results.

FD=*fd* Foreign dataset translation identifier. *fd* is a 3-character code which indicates that foreign dataset translation is to be performed on the dataset. This parameter is required for runtime translation. Valid values for *fd* are:

IBM IBM-compatible sequential file

CDC CDC-compatible sequential file

The default is no translation.

CV=*cv* Foreign dataset conversion mode. CV indicates if implicit data conversion is to be done by the run-time library. CV values are:

- ON Data conversion turned on. ON causes the library to convert the foreign internal representation to or from Cray internal representation, according to the I/O list.

- OFF Data conversion turned off. The data type is not considered when OFF is specified. Full Cray words are moved to or from the foreign dataset.

The default is no data conversion.

CS=*cs* Foreign data character set. This parameter specifies the character set to represent the internal data on the foreign dataset. Run-time library routines convert character data from the *cs* character set to ASCII when implicit data conversion is turned on. The valid *cs* values are:

- AS ASCII

- EB EBCDIC. EB is the default for IBM tape file translation.

- DC Control Data display code. This option is illegal when IBM tape file translation is requested. DC is also the default for CDC tape file translation.

F=*f* Tape format. *f* is a 1- or 2-character code which describes a CDC tape format type. It is required for CDC tape file translation; no default value is provided for F. Valid F values are:

- I Internal tape format
- SI System or SCOPE internal tape format

RF=*rf* Record format, or block and record type. When defined for IBM tape files, RF refers to record format. *rf* is a 1- to 3-character code which describes an IBM record format. Valid values for RF when defining IBM tape files are:

- U Undefined-length records
- F Fixed-length records
- FB Fixed-length blocked records
- V Variable-length records
- VB Variable-length, blocked records
- VBS Variable-length, blocked, spanned records

No default value is provided.

When defined for CDC tape files, RF refers to block and record type. In this case *rf* is a 2-character code which describes a CDC block and record type. The first of the 2-character code describes the block type:

- I Internal block type
- C Character-count block type

The second character of the 2-character code describes the record type:

- W Control-word record type
- Z Zero-byte record type
- S System-logical record type

No default value is provided. RF is required for CDC tape file translation. The following *rf* values are supported for CDC tape files:

- IW Internal block type, control-word record type
- CW Character-count block type, control-word record type
- CZ Character-count block type, zero-byte record type
- CS Character-count block type, system-logical record type

RS=*rs* Tape dataset record size. *rs* is the decimal length of the record, and its expression varies for IBM and CDC tape files.

When defined for IBM tape files, *rs* is the decimal length of the record in 8-bit bytes. The default is set according to the requested record format. Table 8-1 shows the defaults for which RS is set for IBM tape files.

Table 8-1. RS defaults for IBM tape files

<table>
<tr><th>Record format</th><th>Default</th></tr>
<tr><td>Undefined-length</td><td rowspan="3">RS=MBS</td></tr>
<tr><td>Fixed-length</td></tr>
<tr><td>Fixed-length, blocked</td></tr>
<tr><td>Variable-length</td><td rowspan="3">RS≤MBS-4</td></tr>
<tr><td>Variable-length, blocked</td></tr>
<tr><td>Variable-length, blocked, spanned</td></tr>
</table>

In addition, restrictions are enforced on IBM tape files at ASSIGN processing time. Table 8-2 summarizes those restrictions.

Table 8-2. RS restrictions for IBM tape files

Record format	Restriction
Undefined-length	RS=MBS
Fixed-length	
Fixed-length, blocked	RS is multiple of MBS
Variable-length	RS≤MBS-4
Variable-length, blocked	
Variable-length, blocked, spanned	None

For CDC tape files, *rs* is the decimal length of the record in 6-bit characters. *rs* refers to the maximum record length when W is specified as a value for RF. The default or RS=0 imply that there is no maximum record length.

When Z is specified as a value for RF, *rs* becomes the CDC equivalent of the FL parameter: *rs* specifies the length to which zero-byte records are to be extended on input, and the length of a zero-byte record on output. This parameter is required for zero-byte record translation. No default value is provided for *rs* when Z is specified as an RF value.

For CDC system-logical records, *rs* is the maximum record length. The default or RS=0 imply that there is no maximum record length.

MBS=*mbs* Maximum tape block size. *mbs* values are different for IBM and CDC tape files.

When defined for IBM tape files, *mbs* is the maximum block length in 8-bit bytes. The only *mbs* restriction for IBM tape files is that the value be less than or equal to 32760 bytes.

When defined for CDC tape files, *mbs* is the maximum block length in 6-bit characters. The default is D'5120 characters. It is recommended that the user not override this default value.

RELEASE - RELEASE DATASET

The RELEASE control statement relinquishes access to the named datasets for the job. If a dataset is not permanent and its disposition code is SC (scratch), the mass storage assigned to the dataset is released to the system. If the dataset is to be staged, the dataset is entered in the output queue for staging to the destination station. An end-of-data record is written to a permanent dataset and an ADJUST is performed when it is released if the dataset is blocked sequential and the previous operation was a write.

Format:

```
RELEASE,DN=dn1:dn2:...:dn8,HOLD.
```

Parameters:

DN=dn_i	Name of dataset to be released. A maximum of eight datasets may be specified.
HOLD	Hold generic resource. Do not return the resource allocation to the system pool.

INTEGRATED SUPPORT PROCESSOR (ISP) DATASETS

ISP datasets are controlled by two types of COS control statements:

- ISP - initiates communication with the ISP system on behalf of a COS job
- CONNECT - provides access by a COS job to a dataset in the MVS system

See the ISP General Information Manual, CRI publication SG-0094†, for a complete description of these control statements and their use.

† Publication SG-0094 will be available when the ISP software is released.

PERMANENT DATASET MANAGEMENT 9

The permanent dataset management control statements provide methods for creating, protecting, and accessing datasets assigned permanently to mass storage or magnetic tape. Such datasets cannot be destroyed by normal system activity or engineering maintenance.

Permanent dataset management is introduced in section 6. The following permanent dataset management control statements are described in this section:

- SAVE
- ACCESS
- ADJUST
- MODIFY
- DELETE
- PERMIT

SAVE - SAVE PERMANENT DATASET

The SAVE control statement makes a local dataset permanent and defines its associated characteristics for the system. For mass storage datasets, saving involves making an entry in the COS-resident Dataset Catalog (DSC), which uniquely identifies the dataset. For magnetic tape datasets, saving involves front-end servicing to the defined front-end computer system. Under the appropriate conditions, SAVE forces any unwritten data (left in the output buffer) to be written, ensuring that all the data is made permanent. Since this situation occurs when the dataset has been recently written but not yet rewound or closed, SAVE attempts to close the dataset. The specific conditions that the dataset must meet are described under the SAVE macro (see the Macros and Opdefs Reference Manual, CRI publication SR-0012). A permanent dataset is uniquely identified by permanent dataset name (PDN), additional user identification (ID), edition number (ED), and ownership value. SAVE is a system verb.

SAVE has a twofold function:

- Creation of an initial edition of a permanent dataset
- Creation of an additional edition of a permanent dataset

Format:

```
SAVE,DN=dn,PDN=pdn,ID=uid,ED=ed,RT=rt,R=rd,W=wt,M=mn,UQ,NA,ERR,MSG,
EXO={ON|OFF},PAM=mode,ADN=adn(m),TA=opt,TEXT=text,NOTES=notes.
```

Parameters are in keyword form; the only required parameter is DN. Only the DN parameter is valid for tape datasets.

DN=*dn* — Local dataset name. The name the job will use to refer to the dataset while it remains local to the job. This dataset can be closed before the dataset is made permanent.

PDN=*pdn* — Permanent dataset name. The default value is *dn*. The name can be 1 through 15 alphanumeric characters.

ID=*uid* — Additional user identification. 1 through 8 alphanumeric characters assigned by the dataset creator. The default is no user ID.

ED=*ed* — Edition number. A value from 1 through 4095 assigned by the dataset creator. The default value is:

- One, if a permanent dataset with the same PDN and ID does not exist, or
- The current highest edition number plus one, if a permanent dataset with the same PDN and ID does exist.

RT=*rt* — Retention period. User-defined value from 1 through 4095 specifying the number of days a permanent dataset is to be retained by the system. The default value is an installation-defined parameter.

R=*rd* — Read control word. 1 through 8 alphanumeric characters assigned by the dataset creator. The read control word of the highest numbered existing edition of a permanent dataset applies to all subsequent editions of that dataset. The default is no read control word.

W=*wt* — Write control word. 1 through 8 alphanumeric characters assigned by the dataset creator. The write control word of the highest numbered existing edition of a permanent dataset applies to all subsequent editions of that dataset. To obtain write permission, the user must also have unique access (UQ) to that dataset. The default is no write control word.

M=*mn* — Maintenance control word. 1 through 8 alphanumeric characters. The maintenance control word must be specified if a subsequent edition of the same permanent dataset is saved. The default is no maintenance control word.

UQ — Unique access. If the UQ parameter is specified, only this job can access the permanent dataset at the completion of the SAVE function. Otherwise, multiuser access to the permanent dataset is granted.

NA — No abort. If this parameter is omitted, an error causes the job to abort.

ERR — Error message. If this parameter is specified, error termination messages are suppressed.

MSG — Termination message. Normal termination messages are suppressed if MSG is specified.

EXO= $\left\{\begin{matrix}\text{ON}\\ \text{OFF}\end{matrix}\right\}$ — Execute-only dataset. This parameter sets or clears the execute-only status of the dataset. EXO only or EXO=ON causes the dataset to be saved as execute-only. EXO=OFF or omission of this parameter causes the dataset to be saved as nonexecute-only dataset. When EXO=ON has been specified it overides permitted and public access modes.

NOTE

When processing for the SAVE request is complete and EXO=ON, all forms of examination of this dataset are prohibited.

PAM=*mode* Public access mode. The following modes are allowed:

N No public access allowed
E Execute-only
R Read-only
W Write-only
M Maintenance-only

The installation controls the default PAM value.

Combinations of R, W, and M permissions are allowed; for example, PAM=R:W gives both read and write permissions. Note that PAM=E has the same effect as the EXO or EXO=ON parameter and nullifies any other permissions specified.

ADN=*adn* (*m*)
Name of the attributes dataset from which attributes, indicated by the modifiers *m*, are selected. If no modifiers are present, then all attributes are selected. Attribute parameters such as NOTES=, TEXT=, PAM=, R=, etc. take precedence over the modifiers. *adn* must be the local dataset name of a permanent dataset. The modifiers must be enclosed with parentheses and separated by colons. The following modifiers are supported:

Modifier	Selection from attributes dataset
PAM	Public access mode attribute
TRACK	Public access tracking attribute
CW	Control words
PERMITS	Permit list
TEXT	Text attribute
NOTES	Notes attribute
ALL	All attributes

TA=*opt* Track accesses. *opt* can be either YES or NO and indicates whether the owner requires that public accesses to the dataset be tracked. See section 6 for a description of public access and access tracking. The default TA value is NO.

TEXT=*text* Text to be passed to a front-end computer system requesting transfer of the dataset. A maximum of 240 characters can be specified. This text information is considered an attribute of the dataset and is retained along with any other attributes. See section 6 for an explanation of all permanent dataset attributes.

NOTES=*notes*

Notes to be associated with the dataset. A maximum of 480 characters can be specified. There is no restriction on the content of *notes*. A caret symbol in *notes* signifies end-of-line and causes AUDIT to advance to a new line when listing the *notes*. The caret symbol is included in the 480 character maximum limit. *notes* is a permanent dataset attribute. See section 6 for an explanation of all permanent dataset attributes.

ACCESS - ACCESS PERMANENT DATASET

The ACCESS control statement makes an existing permanent dataset local to a job and can be used to create a tape dataset. Following the ACCESS statement, all references to the permanent dataset must be by the local dataset name specified by the DN parameter. ACCESS assures that the user is authorized to use the permanent dataset. The ACCESS control statement must precede the ASSIGN control statement or the request call for the dataset. All tape datasets, whether they are new or not, must be made local via the ACCESS control statement or system request. ACCESS is a system verb.

The user need not access a permanent dataset entered into the System Directory (SDR). A tape dataset cannot reside in the SDR. A basic set of datasets is entered into the System Directory when the operating system is installed. These datasets include the loader, the CFT compiler, the CAL assembler, UPDATE, BUILD, and system utility programs such as copies and dumps (all utilities described in sections 6 through 15 are entered in the System Directory). Other datasets can be entered into the System Directory according to site requirements.

The processing of the ACCESS system request ensures the following:

- The dataset already exists or for new magnetic tape datasets the dataset does not already exist.

- The requested permissions are allowed.

- The type of medium on which the dataset resides has been previously allocated by the job, provided the medium is a dedicated resource (such as magnetic tape).

Format:

ACCESS,DN=*dn*,NA,ERR,MSG,IR,PDN=*pdn*,ID=*uid*,ED=*ed*,R=*rd*,W=*wt*,M=*mn*,UQ,

OWN=*ov*,DT=*dt*,NEW,RING=$\begin{Bmatrix}\text{IN}\\\text{OUT}\end{Bmatrix}$,DEN=*den*,MF=*fes*,

VOL=vol_1:vol_2:...vol_n,FSEC=*fsec*,LB=*lb*,DF=*df*,PROT,MBS=*mbs*,MOD,

XDT=*yyddd*,RT=*rt*,FD=*fd*,CV=*cv*,CS=*cs*,F=*f*,RF=*rf*,RS=*rs*,FSEQ=*fseq*.

Parameters are in keyword form; DN is the only required parameter for mass storage datasets to make an existing permanent dataset local to a job.

The following parameters can be used with mass storage datasets:

DN=*dn* — Local dataset name. The name the job will use to refer to the dataset while it remains local to the job. This is a required parameter.

NA — No abort indicator. This parameter when selected indicates that the job step is not to be aborted if an error arises from the access attempt. If omitted, an error condition causes the job step to be aborted.

ERR — Error message. If this parameter is specified, error termination messages are suppressed.

MSG — Termination message. Normal termination messages are suppressed when MSG is specified.

IR — Immediate reply. An ACCESS request cannot always be honored immediately. When this is the case, the operating system automatically delays the request until it can be honored. IR indicates that control is to return to the caller instead of delaying the request. If IR is specified, the caller has to re-issue the ACCESS request.

PDN=*pdn* Name of a permanent dataset being accessed and already existing in the system. The default value is *dn*. The name can be 1 through 15 characters for mass storage datasets.

ID=*uid* Additional user identification. 1 through 8 alphanumeric characters. If *uid* was specified at SAVE time, the ID parameter must be specified on the ACCESS control statement. The default is no user ID. This parameter applies to mass storage datasets only; it is ignored for magnetic tape datasets.

ED=*ed* Edition number of permanent dataset being accessed; a value from 1 through 4095 was assigned by the dataset creator. If the ED parameter is not specified, the default is the highest edition number known to the system (for this permanent dataset). This parameter applies to mass storage datasets only; it is ignored for magnetic tape datasets.

The following parameters are used to identify the permissions for the accessing of a mass storage permanent dataset.

R=*rd* Read control word as specified at SAVE time. 1 through 8 alphanumeric characters assigned by the dataset creator. The default is no read control word. To obtain read permission, this parameter must be specified on the ACCESS control statement if a read parameter is specified when the dataset is saved. This parameter applies to mass storage datasets only; it is ignored for magnetic tape datasets.

W=*wt* Write control word as specified at SAVE time. To obtain write permission, this parameter must be specified in conjunction with a UQ parameter on the ACCESS control statement if a W parameter is specified when the dataset is saved. Write permission is required for an ADJUST and applies to mass storage datasets only; it is ignored for magnetic tape datasets.

M=*mn* Maintenance control word as specified at SAVE time. This parameter is specified in conjunction with a UQ parameter on an ACCESS control statement if the dataset is to be subsequently deleted. That is, maintenance permission is required to delete a dataset. This parameter applies to mass storage datasets only; it is ignored when used for magnetic tape datasets.

UQ — Unique access. This parameter indicates exclusive access to the dataset is desired. If the UQ parameter is specified and the appropriate write or maintenance control words are specified, then write, maintenance, and/or read permission is granted. If UQ is not specified, then multiuser read access is granted by default (if at a minimum, the read control word is specified). UQ is required to delete a permanent dataset using the DELETE control statement. This parameter applies to mass storage datasets only; it is ignored for magnetic tape datasets.

OWN=*ov* — Ownership value. If the own parameter is specified and the user has been granted access by the owner, the dataset is made local to the job. OWN is ignored if *ov* matches the active ownership value of the job (users need not be permitted to their own datasets).

The following list describes the parameters available for the accessing and/or definition of magnetic tape datasets.

DN=*dn* — Local dataset name. The name the job will use to refer to the dataset while it remains local to the job. This parameter must be present and equated to a valid local dataset name not already in use.

DT=*dt* — Tape dataset generic resource name. This parameter is required for tape datasets. Up to 16 generic resource names can be defined by the installation. Only one generic resource name is available with the released system:

Generic Resource Name	Significance
*TAPE	Device capable of 1600 or 6250 bpi

NEW — Creation disposition. Selection of this parameter indicates the dataset does not yet exist and is to be created by this job. If omitted, the dataset is assumed to already exist. NEW datasets must be written before any read can occur. NEW and MOD are mutually exclusive. NEW automatically selects RING=IN if ring processing is in effect.

RING= $\begin{Bmatrix} \text{IN} \\ \text{OUT} \end{Bmatrix}$ Tape write ring option. The choices are IN if the tape is to be written, and OUT if the tape is to be read only. This parameter is in effect only if the installation parameter I@RNGABT is selected at your site.

DEN=*den* Density of the tape dataset. This parameter applies only to tape datasets; it is ignored when used for mass storage datasets.

6250 Dataset density of 6250 bpi, default
1600 Dataset density of 1600 bpi

MF=*fes* Front-end servicing mainframe identifier. This parameter allows specification of an alternate front-end computer system to which servicing requests are directed. If omitted, the front-end of job origin is used. Front-end servicing is a mechanism whereby auxiliary servicing (such as updating of front-end resident catalogs and tape management systems) of the dataset and/or tape volumes is performed.

The following parameters identify the magnetic tape dataset to be accessed:

PDN=*pdn* Permanent dataset name or file identifier. This parameter can be 1 to 44 characters and is the primary means of identifying the dataset. For labeled tape datasets (AL and SL), the rightmost 17 characters of the PDN are used to match the file identifier from the label group. With front-end servicing the whole value given is generally used as the identifier. If PDN is omitted, then the DN value is used.

VOL=*vol* Volume identifier list. An optional list of 1- through 6-character volume identifiers (VIs) identify tape volumes where the dataset resides. The list contains up to 255 VIs. If the VI list is omitted for a new tape dataset, then the tape volumes on which the dataset is written are selected by the operator and front-end servicing routine. This condition is termed a nonspecific volume allocation. If the VI list is omitted for an old tape dataset, then the volumes on which the dataset resides are determined by front-end servicing. If front-end servicing has no knowledge of the dataset or is inactive, the omission of the VI list results in a job step abort.

FSEC=*fsec* File section number or volume sequence number. This parameter describes on which volume, relative to the first physical volume of the dataset, to begin processing.

The volume sequence number for the first volume of the dataset is 1. If *fsec* is omitted, a value of 1 is assumed. This parameter has a direct relationship to the VIs specified in the VOL parameter. The volume sequence number corresponds to the first VI identified in the VOL parameter. For example, to access a tape dataset starting with the eighth section, specify FSEC=8 on the ACCESS call. If both the MOD and FSEC=*fsec* are coded, the FSEC parameter is not used for validating the header label. Instead it represents the position of the volume serial number in the volume list where MOD processing will begin.

Example: coding the following causes processing to start with tape T2.

```
ACCESS,...MOD,VOL=T1:T2:T3,FSEC=2,...
```

LB=*lb* Tape dataset label type indicating the format of the tape. If this parameter is omitted, label type NL is assumed.

SL	IBM standard labeled tapes
NL	Unlabeled tapes; default.
AL	ANSI standard labeled tapes
FSL	Field format with IBM standard labels
FAL	Field format with ANSII standard labels
FNL	Field format with no labels

Field format tape datasets treat imbedded end-of-files or tapemarks as data. Tapemarks that are not followed by a label are returned in the data EOF control words. On output, EOF control words that are not followed by an EOD control word are converted to physical tapemarks.

The following parameters identify the characteristics of a magnetic tape dataset.

DF=*df* Recording format. This parameter identifies in which format the tape dataset is to be read and/or written. Legal values for this parameter are:

TR	Transparent format
IC	Interchange format

If omitted the format is transparent. For a description of the formats and the associated properties see section 2.

PROT† Front-end protect indicator. This parameter indicates to the front-end computer system performing the service functions that the tape dataset and/or its volumes are to be protected. PROT is recognized for new tape datasets only. If PROT is omitted, the dataset and its volumes are not protected.

MBS=*mbs* Maximum tape block size. If foreign dataset translation is requested by specifying FD, values for *mbs* are different. See the description of the FD parameter below. *mbs* values are different for IBM and CDC tape files.

When defined for IBM tape files, *mbs* is the maximum block length in 8-bit bytes. The only *mbs* restriction for IBM tape files is that the value be less than or equal to 32760 bytes.

When defined for CDC tape files, *mbs* is the maximum block length in 6-bit characters. The default is D'5120 characters. It is recommended that the user not override this default value to ensure interchangeability with all CYBER operating systems.

If MBS is omitted and the dataset is new, a default size determined by the site is used. The limiting value of the parameter is also left to site definition. If omitted for an existing labeled dataset (AL or SL), the maximum block size is set to the value from the label group. Exceeding this size when writing results in a job abort condition of WRITE FORMAT ERROR. When reading a tape block that is larger than the specified value, a job abort condition of LARGE BLOCK ENCOUNTERED is produced. MBS is rounded up to the next multiple of 4096 bytes for transparent format tape datasets.

MOD Existing tape dataset modification identifier. This parameter allows the user to position single volume and multivolume datasets on tape. It specifies that data will be added at the end of an existing dataset on either standard labeled or unlabeled tapes. Access requests using MOD for tape volume positioning are only successful if the end of a dataset is indicated by the end-of-file trailer label for a labeled tape volume, and by a tape mark for an unlabeled tape. MOD and NEW are mutually exclusive. MOD selects RING=IN if ring processing is in effect.

---

† Deferred implementation

XDT=*yyddd* Expiration date. Indicates the date this tape dataset may be overwritten. *yy* specifies the year and is a number from 0 through 99. *ddd* specifies the day in theyear and is a number from 001 through 366. This parameter identifies the year and the day on which a new tape dataset is considered dormant. If omitted and the dataset is going to be written, the current date is used. This parameter is also used as a means of communicating with a servicing front-end computer system. The XDT and RT parameters are mutually exclusive.

RT=*rt* Retention period. User-defined value from 1 through 4095 specifying the number of days a permanent dataset is to be retained by the system. The RT parameter is similar to the XDT parameter but allows the user to specify relative expiration date. If RT is omitted, the default value used is no days of retention. This parameter is mutually exclusive with the XDT parameter.

The following tape dataset parameters specify that record and data format conversion are to be performed on the tape dataset at run time.

FD=*fd* Foreign dataset translation identifier. *fd* is a 3-character code which indicates that foreign dataset translation is to be performed on the dataset. This parameter is required for run-time translation. Valid values for *fd* are:

IBM IBM-compatible sequential file
CDC CDC-compatible sequential file

The default is no translation.

CV=*cv* Foreign dataset conversion mode. CV indicates if implicit data conversion is to be done by the runtime library. CV values are:

ON Data conversion turned on. ON causes the library to convert the foreign internal representation to or from Cray internal representation, according to the I/O list.
OFF Data conversion turned off. The data type is not considered when OFF is specified. Full Cray words are moved to or from the foreign dataset.

The default is no data conversion.

CS=*cs* Foreign data character set. This parameter specifies the character set to represent the internal data on the foreign dataset. Run-time library routines convert character data from the *cs* character set to ASCII when implicit data conversion is turned on.

The valid *cs* values are:

AS ASCII
EB EBCDIC. EB is the default for IBM tape file translation.
DC Control Data display code. This option is illegal when IBM tape file translation is requested. DC is also the default for CDC tape file translation.

F=*f* Tape format. *f* is a 1- or 2-character code which describes a CDC tape format type. It is required for CDC tape file translation. No default value is provided for F. Valid F values are:

I Internal tape format
SI System or SCOPE internal tape format

RF=*rf* Record format, or block and record type. When defined for IBM tape files, RF refers to record format. *rf* is a 1- to 3-character code which describes an IBM record format. Valid values for RF when defining IBM tape files are:

U Undefined-length records
F Fixed-length records
FB Fixed-length blocked records
V Variable-length records
VB Variable-length, blocked records
VBS Variable-length, blocked, spanned records

No default value is provided. However, RF can be omitted when accessing an IBM standard labeled tape file. In that case the record format designated on the label is used. If NEW is specified, RF=U.

When defined for CDC tape files, RF refers to block and record type. In this case *rf* is a 2-character code which describes a CDC block and record type. The first of the 2-character code describes the block type:

I Internal block type
C Character-count block type

The second character of the 2-character code describes the record type:

W Control-word record type
Z Zero-byte record type
S System-logical record type

No default value is provided. RF is required for CDC tape file translation. The following *rf* values are supported for CDC tape files:

IW Internal block type, control-word record type
CW Character-count block type, control-word record type
CZ Character-count block type, zero-byte record type
CS Character-count block type, system-logical record type

RS=*rs* Tape dataset record size. *rs* is the decimal length of the record, and its expression varies for IBM and CDC tape files.

When defined for IBM tape files, *rs* is the decimal length of the record in 8-bit bytes. The default is set according to the requested record format. However, no default value is used when accessing an IBM standard labeled tape file. Instead the record size designated by the label is used. Table 9-1 shows the defaults for which RS is set for IBM tape files.

Table 9-1. RS defaults for IBM tape files

Record format	Default
Undefined-length	RS=MBS
Fixed-length	
Fixed-length, blocked	
Variable-length	RS=MBS-4
Variable-length, blocked	
Variable-length, blocked, spanned	

In addition, restrictions are enforced on IBM tape files at ACCESS processing time. Table 9-2 summarizes those restrictions. Nonetheless, restrictions are not enforced if the tape file accessed is an IBM standard labeled tape file, and if neither RS nor MBS are specified.

Table 9-2. RS restrictions for IBM tape files

<table>
<tr><th>Record format</th><th>Restriction</th></tr>
<tr><td>Undefined-length</td><td rowspan="2">RS=MBS</td></tr>
<tr><td>Fixed-length</td></tr>
<tr><td>Fixed-length, blocked</td><td>MBS is multiple of RS</td></tr>
<tr><td>Variable-length</td><td rowspan="2">RS≤MBS-4</td></tr>
<tr><td>Variable-length, blocked</td></tr>
<tr><td>Variable-length, blocked, spanned</td><td>None</td></tr>
</table>

For CDC tape files, *rs* is the decimal length of the record in 6-bit characters.

rs refers to the maximum record length when W is specified as a value for RF. The default or RS=0 imply that there is no maximum record length.

When Z is specified as a value for RF, *rs* becomes the equivalent of the CDC FL parameter: *rs* specifies the length to which zero-byte records are to be extended with blank characters on input and the length of a zero-byte record on output. This parameter is required for zero-byte record translation. No default value is provided for *rs* when Z is specified as an RF value.

For CDC system-logical records, *rs* is the maximum record length. The default or RS=0 imply that there is no maximum record length.

FSEQ=*fseq* File sequence number. This is a one- to four-digit number that describes the relative position of the dataset on the tape volume. The default is 1.

ADJUST - ADJUST PERMANENT DATASET

The ADJUST control statement changes the size of a mass storage permanent dataset; that is, it redefines the size of the dataset. When a permanent dataset is overwritten, and the dataset size changes, issuing an ADJUST statement informs the system of the dataset's new size. An ADJUST of a permanent dataset can be issued if the dataset has been previously accessed within the job with write permission. ADJUST is a system verb.

Under the appropriate conditions, ADJUST forces any unwritten data to mass storage to ensure that all of the dataset is made permanent. Since this situation occurs when the dataset has been recently written to but not yet closed, ADJUST attempts to close the dataset. The specific conditions that the dataset must meet are described under the ADJUST macro (see the Macros and Opdefs Reference Manual, CRI publication SR-0012).

The ADJUST statement is ignored when used with magnetic tape datasets.

Format:

```
ADJUST,DN=dn,NA,ERR,MSG.
```

Parameters:

DN=*dn*	Local dataset name of a permanent dataset that has been accessed with write permission. This dataset can be closed before the ADJUST statement is processed.
NA	No abort. If this parameter is omitted, an error causes the job step to abort.
ERR	Error message. If this parameter is specified, error termination messages are suppressed.
MSG	Termination message. Normal termination messages are suppressed when MSG is specified.

MODIFY - MODIFY PERMANENT DATASET

The MODIFY control statement changes permanent dataset information established by the SAVE function or a previously executed MODIFY function. A permanent dataset must be accessed with unique access (UQ) and all permissions before MODIFY can be issued. MODIFY is a system verb.

Once a permanent dataset exists, the read, write, and maintenance control words, public access mode, and access tracking may apply to subsequent editions of that permanent dataset. MODIFY applies to mass storage datasets only; it is ignored for tape datasets.

Format:

```
MODIFY,DN=dn,PDN=pdn,ID=uid,ED=ed,RT=rt,R=rd,W=wt,M=mn,NA,ERR,MSG,

    EXO={ON|OFF},PAM=mode,TA=opt,TEXT=text,NOTES=notes.
```

Parameters are in keyword form; the only required parameter is DN.

DN=*dn* — Local dataset name of a permanent dataset that has been accessed with all permissions. DN is a required parameter.

PDN=*pdn* — New permanent dataset name to be applied to the existing dataset. If this parameter is omitted, the existing permanent dataset name is retained.

ID=*uid* — New additional user identification, to be applied to the existing permanent dataset. 1 through 8 alphanumeric characters. If this parameter is omitted, the existing user ID is retained. If this parameter is present without a value, user identification is established as binary zeros.

ED=*ed* — New edition number to be applied to the existing permanent dataset. If this parameter is omitted, the existing edition number is retained.

RT=*rt* — New retention period to be applied to the existing permanent dataset. If this parameter is omitted, the current retention period is retained. If this parameter is present without a value, the retention period is set to the installation-defined value.

R=*rd* — New read permission control word to be applied to the existing permanent dataset. If this parameter is omitted, the existing read permission is retained. If R is present without a value, read permission is established as binary zeros.

W=*wt* — New write permission control word to be applied to the existing permanent dataset. If this parameter is omitted, the existing write permission is retained. If W is present without a value, write permission is established as binary zeros.

M=*mn* New maintenance permission control word to be applied to the existing permanent dataset. If this parameter is omitted, the existing maintenance permission is retained. If M is present without a value, maintenance permission is established as binary zeros.

NA No abort. If this parameter is omitted, an error causes the job to abort.

ERR Error message. If this parameter is specified, error termination messages are suppressed.

MSG Termination message. Normal termination messages are suppressed when MSG is specified.

EXO= $\left\{\begin{matrix}\text{ON}\\ \text{OFF}\end{matrix}\right\}$ Execute-only dataset. This parameter sets or clears the execute-only status of a dataset. EXO only or EXO=ON causes the dataset to be modified to execute-only. EXO=OFF causes the dataset to be modified to a nonexecute-only dataset. If this parameter is omitted, the execute-only status of a dataset is unchanged.

NOTE

When processing for the MODIFY request is complete and EXO=ON, all forms of examination of this dataset are prohibited.

PAM=*mode* Public access mode. The following modes are allowed:

- N No public access allowed
- E Execute only
- R Read only
- W Write only
- M Maintenance only

The installation controls the default PAM value. Combinations of R, W, and M permissions are allowed; for example, PAM=R:W gives both read and write permissions. Note that PAM=E has the same effect as the EXO or EXO=ON parameter and nullifies any other permissions specified.

TA=*opt* Track accesses. *opt* can be either YES or NO and indicates whether the owner requires that public accesses to the dataset be tracked. See section 6 for a description of public access and access tracking. The default TA value is NO.

TEXT=*text* Text to be passed to a front-end computer system requesting transfer of the dataset. A maximum of 240 characters can be specified. This text information is considered an attribute of the dataset and is retained along with any other attributes. See section 6 for an explanation of all permanent dataset attributes.

NOTES=*notes*
Notes to be associated with the dataset. A maximum of 480 characters can be specified. There is no restriction on the content of *notes*. A caret symbol in *notes* signifies end-of-line and causes AUDIT to advance to a new line when listing the *notes*. The caret symbol is included in the 480 character maximum limit. *notes* is a permanent dataset attribute. See section 6 for an explanation of all permanent dataset attributes.

DELETE - DELETE PERMANENT DATASET

The DELETE control statement clears the permanence state for a dataset. For mass storage datasets this involves clearing the dataset's definition from the Dataset Catalog (DSC). For magnetic tape datasets, a request to remove the dataset's definition from the front-end's catalog is sent to the servicing front-end computer system. If PARTIAL is specified, the dataset is deleted but its attributes are retained. To issue a DELETE of a dataset, the job must have previously accessed the dataset with maintenance permission, if a maintenance control word exists for the dataset, and unique access (UQ). The dataset remains a local dataset after deletion until job termination or execution of a RELEASE control statement. DELETE is a system verb.

Format:

```
DELETE,DN=dn,NA,ERR,MSG,PARTIAL.
```

Parameters:

DN=*dn* Local dataset name of a permanent dataset accessed with maintenance permission and unique access

NA No abort. If this parameter is omitted, a fatal error causes the job step to abort.

ERR Error message. If this parameter is specified, error termination messages are suppressed.

MSG Termination message. Normal termination messages are suppressed if MSG is specified.

PARTIAL Partial delete. Presence of this keyword causes the system to delete only the mass storage resident data. The DSC entry and the dataset's attributes information are retained. PARTIAL can be specified only for a mass storage dataset.

PERMIT - EXPLICITLY CONTROL ACCESS TO DATASET

The PERMIT control statement allows a user to explicitly designate who can access a particular permanent dataset. PERMIT applies to all editions of the permanent dataset. This dataset need not be local for PERMIT to be executed. PERMIT applies to user permanent mass storage datasets only. Access permission given with a PERMIT control statement takes precedence over the PAM parameter described under SAVE and MODIFY. PERMIT is a system verb.

Format:

PERMIT,PDN=*pdn*,ID=*uid*,AM=*m*,RP,USER=*ov*,ADN=*adn*,NA,ERR,MSG.

Parameters:

PDN=*pdn* Name of an existing user permanent dataset. The name can be 1 through 15 characters. PDN is a required parameter.

ID=*uid* Additional user identification. 1 through 8 alphanumeric characters. If ID was specified on the SAVE request, the ID parameter must be specified on the PERMIT control statement. The default is no user ID.

AM=*m* Access mode permitted for alternate user. These modes are:

N No dataset access allowed
E Execute-only
R Read-only
W Write-only
M Maintenance-only

Each installation controls the default AM value. Combinations of R, W, and M permissions are allowed; for example, AM=R:W gives both read and write permissions. Note that AM=E gives the permitted user execute-only access to the dataset, effectively nullifying any other permissions specified.

RP — Remove permit parameter. Removes the permit associated with the specified ownership value.

USER=*ov* — User ownership value associated with the user being permitted

ADN=*adn* — Local dataset name of the attributes dataset from which the permit list is copied

NA — No abort. If this parameter is omitted, an error causes the job step to abort.

ERR — Error message. If this parameter is specified, error termination messages are suppressed.

MSG — Termination message. Normal termination messages are suppressed when MSG is specified.

EXAMPLES OF PERMANENT DATASET CONTROL STATEMENTS

To clarify the permanent dataset management control statements, some examples follow:

1. A user identified as USERXYZ creates a permanent dataset, which no other user can access. All subsequent editions of this dataset share this attribute.

   ```
   SAVE,DN=ABC,PDN=EXAMPLE1,ED=1,PAM=N,TA=NO.
   ```

2. A user identified as USERXYZ, creates a permanent dataset, which can be accessed by all other users in read mode.

   ```
   SAVE,DN=XYZ,PDN=EXAMPLE2,ED=1,PAM=R,TA=NO.
   ```

3. An alternate user is accessing the permanent dataset created in example 2.

   ```
   ACCESS,DN=LOCAL,PDN=EXAMPLE2,ED=1,OWN=USERXYZ.
   ```

   The system does not track the alternate user access since the dataset was created with TA=NO.

4. Allow another user (known in this example as USER1) to access the permanent dataset created in example 1 in read and execute mode only.

```
PERMIT,PDN=EXAMPLE1,USER=USER1,AM=R:E.
```

5. Enable public access tracking for the permanent dataset created in example 2.

```
ACCESS,DN=LOCAL,PDN=EXAMPLE2,ED=1,UQ.
MODIFY,DN=LOCAL,TA=YES.
```

6. Permit write mode access for PDN=EXAMPLE2 to users known as USER2 and USER3.

```
PERMIT,PDN=EXAMPLE2,USER=USER2,AM=W.
PERMIT,PDN=EXAMPLE2,USER=USER3,AM=W.
```

7. Change the permission granted to USER1 in example 4 to AM=W.

```
PERMIT,PDN=EXAMPLE1,USER=USER1,AM=W.
```

8. Remove the access permission granted to USER1 in example 7.

```
PERMIT,PDN=EXAMPLE1,USER=USER1,RP.
```

9. User USERXYZ acquires a dataset, then permits another user to use it and subsequently partially deletes the dataset to retain just the PERMITs and TEXT information.

```
ACQUIRE,DN=EX9,TEXT='..........',UQ.
PERMIT ,PDN=EX9,USER=SOMEONE,AM=R.
DELETE ,DN=EX9,PARTIAL.
```

10. User USERXYZ creates a permits template.

```
SAVE,DN=EX10,PDN=PERMS,^
  NOTES='PERMITS TEMPLATE FOR AERO USERS.  ' ^
  'THESE PERMITS SHOULD BE REMOVED AFTER OCT 31, 1983.',UQ.
PERMIT,PDN=PERMS,USER=USERA,AM=E.
PERMIT,PDN=PERMS,USER=USERB,AM=R.
PERMIT,PDN=PERMS,USER=USERC,AM=W.
DELETE,DN=EX10,PARTIAL.
```

11. User SOMEONE acquires the dataset that was partially deleted in example 9.

```
ACQUIRE,DN=LOCAL,PDN=EX9,OWN=USERXYZ.
```

Note that the TEXT need not be specified and that after the dataset has been acquired from the front-end computer system, it is made permanent and belongs to user USERXYZ.

DATASET STAGING CONTROL 10

Staging is the process of transferring jobs and data in the form of COS datasets from a front-end computer system to Cray mass storage or of transferring datasets from Cray mass storage to a front-end computer system. Dataset staging control is introduced in section 6.

Three control statements support staging datasets between Cray mass storage and a front-end system: ACQUIRE, FETCH, and DISPOSE. Another control statement, SUBMIT, directs datasets to the COS input queue.

ACQUIRE - ACQUIRE PERMANENT DATASET

The ACQUIRE control statement allows the user to make a dataset permanent and accessible to the job making the request. ACQUIRE is a system verb. Some ACQUIRE control statement examples are included with the permanent dataset management examples (see section 10).

When an ACQUIRE control statement is issued, COS determines if the requested dataset is front-end resident or permanently resident on Cray mass storage by checking the COS Permanent Dataset Catalog (DSC) for a dataset with matching PDN, ID, ED, and ownership value fields.

If COS determines that the requested dataset is already permanently resident on Cray mass storage, dataset access is granted to the job making the request.

If the requested dataset is not a COS mass storage permanent dataset, the request for the dataset is sent to the front-end system. The front-end system stages the dataset to Cray mass storage. COS then makes the dataset permanent and grants dataset access to the job making the request. Until the dataset is made permanent, processing of the job making the request is delayed.

Format:

```
ACQUIRE,DN=dn,PDN=pdn,ID=uid,ED=ed,RT=rt,R=rd,W=wt,M=mn,UQ,

TEXT=text,MF=mf,TID=tid,DF=df,OWN=own,PAM=mode,

ADN=adn(m),TA=opt,NOTES=notes,ERR,MSG.
```

Parameters are in keyword form; the only required parameter is DN.

DN=*dn* — Local dataset name. The name the job will use to refer to the dataset while it remains local to the job. 1 through 7 alphanumeric characters, the first of which is A through Z, $, @, or %; remaining characters can also be numeric. DN is a required parameter.

PDN=*pdn* — Name of COS permanent dataset to be accessed or staged from a front-end system, saved, and accessed. The permanent dataset name is passed to the front-end system; it is the name saved by the system if the dataset is staged. *pdn* is 1 through 15 alphanumeric characters assigned by the dataset creator. The default for *pdn* is *dn*.

ID=*uid* — Additional user identification. 1 through 8 alphanumeric characters assigned by the dataset creator. The default is no user ID.

ED=*ed* — Edition number. A value from 1 through 4095 assigned by the dataset creator. The default value is:

- One, if a permanent dataset with the same PDN and ID does not currently exist, or
- The current highest edition number of that dataset if the permanent dataset with the specified PDN and ID does exist.

RT=*rt* — Retention period. User-defined value from 1 through 4095 specifying the number of days a permanent dataset is to be retained by the system. The default value is an installation-defined parameter.

R=*rd* Read control word. 1 through 8 alphanumeric characters assigned by the dataset creator. The default is no read control word.

W=*wt* Write control word. 1 through 8 alphanumeric characters assigned by the dataset creator. The default is no write control word.

M=*mn* Maintenance control word. 1 through 8 alphanumeric characters assigned by the dataset creator. The control word must be specified if a subsequent edition of the permanent dataset is saved. If no staging occurs, and the dataset is to be deleted, this parameter can be specified in conjunction with the UQ parameter (that is, maintenance permission is required to delete a dataset).

UQ Unique access. If the UQ parameter is specified, the job is granted unique access to the permanent dataset; otherwise, multiaccess to the permanent dataset is granted. If no staging is performed because the dataset already exists, write, maintenance, and/or read permission can be granted if the appropriate read, write, and/or maintenance control words are specified.

TEXT=*text* Text to be passed to a front-end computer system requesting transfer of the dataset. A maximum of 240 characters can be specified. This text information is considered an attribute of the dataset and is retained along with any other attributes. See section 6 for an explanation of all permanent dataset attributes.

MF=*mf* Identifier for the front-end computer. 2 alphanumeric characters. The default is the front end of job origin.

TID=*tid* Terminal identifier. 1 through 8 alphanumeric characters identifying destination terminal. The default is the terminal of job origin.

DF=*df* Dataset format. This parameter defines whether a dataset is to be presented to the Cray Computer System in COS blocked format and whether the front-end system is to perform character conversion. The default is CB.

For example, a user wishes to acquire a dataset from magnetic tape in blocked binary as it appears at the front-end system. In this case, BB is specified.

df is a 2-character alphanumeric code defined for use on the front-end system. Cray Research, Inc., suggests support of the following codes:

CD Character deblocked. The front-end system performs character conversion to 8-bit ASCII, if necessary.

CB Character blocked. The front-end system blocks the dataset before staging and performs character conversion to 8-bit ASCII, if necessary.

BD Binary deblocked. The front-end system does not perform character conversion. For ACQUIRE, BD is the same as TR.

BB Binary blocked. The front-end system blocks the dataset before staging but does not do character conversion.

TR Transparent. No blocking/deblocking or character conversion is performed.

OWN=*ov* Ownership value. If the own parameter is specified and the user has been granted access by the owner, the dataset is made local to the job. OWN is ignored if *ov* matches the active ownership value of the job (users need not be permitted to their own datasets).

PAM=*mode* Public access mode. The following modes are allowed:

N No public access allowed
E Execute only
R Read only
W Write only
M Maintenance only

Combinations of R, W, and M permissions are allowed; for example, PAM=R:W gives both read and write permissions. Note that PAM=E has the same effect as the EXO or EXO=ON parameter and nullifies any other permissions specified. Each installation controls the default PAM value.

ADN=*adn*(*m*)

Name of attributes dataset from which attributes, indicated by the modifiers *m*, are selected. If no modifiers are present, then all attributes are selected. Attribute parameters such as NOTES=, TEXT=, PAM=, R=, etc. take precedence over the modifiers. *adn* must be the local dataset name of a permanent dataset. The modifiers must be enclosed with parentheses and separated by colons. The following modifiers are supported:

Modifier	Selection from attributes dataset
PAM	Public access mode attribute
TRACK	Public access tracking attribute
CW	Control words
PERMITS	Permit list
TEXT	Text attribute
NOTES	Notes attribute
ALL	All attributes

TA=*opt* Track accesses. *opt* can be either YES or NO and indicates whether the owner requires that public accesses to the dataset be tracked. See section 6 for a description of public access and access tracking. The default TA value is NO.

NOTES=*notes*

Notes to be associated with the dataset. A maximum of 480 characters can be specified. There is no restriction on the content of *notes*. A caret symbol in *notes* signifies end-of-line and causes AUDIT to advance to a new line when listing the *notes*. The caret symbol is included in the 480 character maximum limit. *notes* is a permanent dataset attribute. See section 6 for an explanation of all permanent dataset attributes.

ERR Error message. If this parameter is specified, error termination messages are suppressed.

MSG Termination message. Normal termination messages are suppresed when MSG is specified.

DISPOSE - DISPOSE DATASET

The DISPOSE control statement directs a dataset to the COS output queue for staging to a specified front-end computer system. DISPOSE can also be used to alter the effects of a previous DISPOSE,DEFER of the same dataset.

Defining the DISPOSE characteristics can be done before the actual staging via the DEFER parameter. The DEFER parameter saves all selected dispose parameters for use when the dataset is released, which is when the actual staging is initiated. DISPOSE is a system verb.

Format:

```
DISPOSE,DN=dn,SDN=sdn,DC=dc,DF=df,MF=mf,SF=sf,ID=uid,TID=tid,
    ED=ed,RT=rt,R=rd,W=wt,M=mn,TEXT=text,WAIT,NOWAIT,DEFER,NRLS.
```

Parameters are in keyword form; the only required parameter is DN.

DN=*dn* Local dataset name. Name by which the dataset is known to the user job. DN is a required parameter.

SDN=*sdn* Staged dataset name. 1 through 15 character name by which the dataset is to be known at the destination front end. The default for *sdn* is *dn*.

DC=*dc* Disposition code. Disposition to be made of the dataset. If the DC parameter is omitted, the default is PR.

dc is a 2-character alphanumeric code describing the destination of the dataset as follows:

IN Input (job) dataset. Dataset is queued as a job on the mainframe specified with the MF parameter.

ST Stage to front end. Dataset is made permanent at the front end designated by the MF parameter.

SC Scratch dataset. Dataset is released, unless another DISPOSE request is still pending on the dataset. This parameter has the same effect as RELEASE,DN=*dn*.

PR Print dataset. Dataset is printed on a printer available at the front end designated by the MF parameter.

PU Punch dataset. Dataset is punched on any card punch available at the front end designated by the MF parameter.

PT Plot dataset. Dataset is plotted on any available plotter at the front end designated by the MF parameter.

MT Write dataset on magnetic tape at the front end designated by the MF parameter.

NOTE

The dataset dispositions noted above are by convention only. Actual dataset disposition is determined by the destination front end.

DF=*df* Dataset format. This parameter defines whether a dataset is sent from the Cray Computer System in COS-blocked format and whether the front-end system is to perform character conversion. The default is CB.

For example, a user wishes to save a dataset on magnetic tape in blocked binary as it appears on COS mass storage. In this case, BB is specified. A user who wants a dataset printed can specify CB if the front-end computer handles deblocking.

df is a 2-character alphanumeric code defined for use on the front-end system. Cray Research, Inc., suggests support of the following codes:

CD Character deblocked. The front-end system performs character conversion from 8-bit ASCII, if necessary.

CB Character blocked. No deblocking is performed at the Cray mainframe before staging. The front-end system performs deblocking and character conversion from 8-bit ASCII, if necessary.

BD Binary deblocked. The front-end system does not perform character conversion. For DISPOSE, BD is the same as TR.

BB Binary blocked. The front-end system does not perform character conversion. The Cray mainframe does not perform deblocking before staging. The front-end system is expected to perform deblocking.

TR Transparent. No blocking/deblocking or character conversion is performed.

Other codes can be added by the local site. Undefined pairs of characters can be passed but are treated as transparent mode by COS.

MF=*mf* Front-end computer identifier. 2 alphanumeric characters. Identifies the front end to which the dataset is to be staged. If omitted, the front end where the issuing job originated is used. If MF is given a value of a Cray mainframe ID and DC=IN, an error message is issued and the job step is aborted (see the SUBMIT control statement later in this section).

SF=*sf* Special form information to be passed to the front-end system. 1 through 8 alphanumeric characters. SF is defined by the needs of the front-end system.

ID=*uid* Additional user identification. 1 through 8 alphanumeric characters assigned by the dataset creator. The default is no user ID.

TID=*tid* Terminal identifier. 1 through 8 alphanumeric characters identifying destination terminal. The default is terminal of job origin, where applicable.

ED=*ed* Edition number, meaningful only if DC=ST. A user-defined value from 1 through 4095. The default value depends on the destination front end.

RT=*rt* Retention period, meaningful only if DC=ST. A user-defined value from 1 through 4095 specifying the number of days a dataset is to be retained by the destination front end. The default value depends on the destination front end.

R=*rd* Read control word, meaningful only if DC=ST. 1 through 8 alphanumeric characters. The default is no read control word.

W=*wt* Write control word, meaningful only if DC=ST. 1 through 8 alphanumeric characters. The default is no write control word.

M=*mn* Maintenance control word, meaningful only if DC=ST. 1 through 8 alphanumeric characters. The default is no maintenance control word.

TEXT=*text* Text to be passed to the front-end system requesting transfer of a dataset. The format for TEXT is defined by the front-end system for managing its own datasets or files. Typically, *text* is in the form of one or more control statements for the front-end system; these statements must contain their own terminator for the front end. *text* cannot exceed 240 characters.

NOTE

text specified on the DISPOSE control statement is not the same as the permanent dataset *text* attribute. Any *text* existing as a permanent dataset attribute is ignored by DISPOSE (see section 6 for discussion).

WAIT Job wait. When this parameter is specified, the job does not resume processing until the disposed dataset has been staged to the front-end system. If the front-end system cancels the transfer, the waiting job is aborted and job step abort processing occurs as described in section 3. If WAIT is not specified, processing can resume immediately upon issue of the DISPOSE, depending upon an installation option. The WAIT parameter is useful in detecting unsuccessful transfers.

NOWAIT When this parameter is specified, the job does not wait until the dataset has been staged to the front-end system but resumes processing immediately. If the front-end system cancels the transfer, no special action is taken; that is, the job is not aborted. If neither WAIT or NOWAIT are specified, processing can resume immediately upon issue of the DISPOSE, depending upon an installation option.

DEFER When this parameter is specified, the disposition occurs when the dataset is released either by a RELEASE request or job termination. The dispose characteristics are saved and used when the dataset is released.

NRLS No release. When this parameter is specified, the dataset remains local to the job after the DISPOSE request has been processed. When NRLS is specified on a DISPOSE control statement, the dataset cannot be written to, until the transfer to the specified front end is completed. Therefore, it is advisable to use WAIT with NRLS.

SUBMIT - SUBMIT JOB DATASET

With SUBMIT, a job running on the Cray mainframe can direct another dataset (which must also be a job) to the COS input queue. The job that is submitted executes independently of the submitting job. SUBMIT is a system verb.

Format:

```
SUBMIT,DN=dn,SID=sf,DID=df,TID=tid,DEFER,NRLS.
```

Parameters are in keyword format; the only required parameter is DN.

DN=*dn* Local dataset name. A valid local dataset name. DN is a required parameter and must be given a value.

SID=*sf* Default source front-end system identifier; 2 alphanumeric characters. If an MF parameter is not specified in an ACQUIRE or FETCH control statement within the submitted job, the SID parameter defines the default source front-end system for the dataset to be acquired. If the MF and SID parameters are omitted, the default source identifier of the submitting job is used.

DID=*df* Default destination front-end identifier; 2 alphanumeric characters. If an MF parameter is not specified in a DISPOSE control statement within the submitted job, the DID parameter defines the default destination front-end system for the dataset to be disposed. If the MF and DID parameters are omitted, the default destination identifier of the submitting job is used.

TID=*tid* Default terminal identifier. 1 through 8 alphanumeric character identifier defining the default terminal ID for the submitted job. If TID is omitted, then the terminal ID of the submitting job is used.

DEFER Deferred submit. Selection of this parameter causes the SUBMIT characteristics to be defined, with a release of the dataset actually initiating the submit of the dataset. If DEFER is omitted, the SUBMIT occurs immediately.

NRLS No release. This parameter indicates if the dataset is to remain local to the job after SUBMIT has been processed. If NRLS is omitted, the dataset is released after the SUBMIT. If selected, the dataset remains local to the job after the SUBMIT and is available for reading only.

FETCH - FETCH LOCAL DATASET

The FETCH control statement allows the user to make a dataset that resides on a front-end computer system local to the COS job. The dataset is transferred from the front-end computer system. The dataset is not made permanent on the Cray Computer System. The originating job is delayed until the dataset arrives on Cray mass storage.

Format:

FETCH,DN=*dn*,SDN=*sdn*,TEXT=*text*,MF=*mf*,TID=*tid*,DF=*df*.

Parameters are in keyword form; the only required parameter is DN.

DN=*dn* Local dataset name. The name the job will use to refer to the dataset while it remains local to the job. 1 through 7 alphanumeric characters, the first of which is A through Z, $, @, or %; remaining characters can also be numeric. DN is a required parameter.

SDN=*sdn* Staged dataset name. 1 through 15 alphanumeric characters. Name by which the dataset is known on the front end. The default for *sdn* is *dn*.

DF=*df* Dataset format. This parameter defines whether a dataset is sent from the Cray Computer System in COS blocked format and whether the front-end system is to perform character conversion. The default is CB.

For example, a user wishes to save a dataset on magnetic tape in blocked binary as it appears on COS mass storage. In this case, BB is specified. A user who wants a dataset printed can specify CB if the front-end computer handles deblocking.

df is a 2-character alphanumeric code defined for use on the front-end system. Cray Research, Inc., suggests support of the following codes:

CD Character deblocked. The front-end system performs character conversion to 8-bit ASCII, if necessary.

CB Character blocked. The front-end system blocks the dataset before staging and performs character conversion to 8-bit ASCII, if necessary.

BD Binary deblocked. The front-end system does not perform character conversion. For FETCH, BD is the same as TR.

BB Binary blocked. The front-end system blocks the dataset before staging but does not do character conversion.

TR Transparent. No blocking/deblocking or character conversion is performed.

Other codes can be added by the local site. Undefined pairs of characters can be passed but are treated as transparent mode by COS.

MF=*mf* Mainframe computer identifier. 2 alphanumeric characters. The default is the front end of job origin.

TID=*tid* Terminal identifier. 1 through 8 characters identifying destination terminal. The default is terminal of job origin where applicable.

TEXT=*text* Text to be passed to the front-end system requesting transfer of a dataset. The format for TEXT is defined by the front-end system for managing its own datasets or files. Typically, *text* is in the form of one or more control statements for the front-end system; these statements must contain their own terminator for the front end. *text* cannot exceed 240 characters.

PERMANENT DATASET UTILITIES 11

The following utility routines support permanent datasets:

- PDSDUMP dumps all specified permanent datasets to a user-specified dataset. Input and output datasets can be included in the dump.
- PDSLOAD loads permanent datasets that have been dumped by PDSDUMP and updates or regenerates the Dataset Catalog. Input and output datasets are also loaded through PDSLOAD.
- AUDIT produces a report containing status information for each permanent dataset. AUDIT does not include input or output datasets.

All of the permanent dataset utilities permit a shorthand notation for the arguments to the PDN (or PDS), ID, US, and OWN parameters. Using this notatation, a dash represents any number of characters or no characters and an asterisk represents any one character.

Examples:

PDN=ABC-	List all permanent dataset names beginning with ABC.
PDN=A***	List all 4-character permanent dataset names beginning with A.
PDN=-A*-	List all permanent dataset names containing the letter A followed by one or more other characters.
PDN=-	List all permanent dataset names.
PDN=***-	List all permanent dataset names having three or more characters.

When permanent dataset privacy is enabled, callers of these utilities are limited to actions on their own datasets unless the CW parameter is present on the control card. The OWN and NOWN parameters cannot be specified unless CW is also specified. When privacy is enabled, the US value from the JOB or ACCOUNT control statement is an implied dataset selection criterion, unless the CW parameter is present. When privacy is not enabled, the US value from the JOB or ACCOUNT control statement is not used as a selection criterion. CW must be specified if US is specified on the permanent dataset utility control statement.

PDSDUMP - DUMP PERMANENT DATASETS

PDSDUMP dumps specified permanent datasets to a dataset that can be saved or staged to a station as desired. Characteristics and conditions that cause a dataset to be omitted from dumping include:

- Execute-only dataset
- Dataset allocation conflict
- Catastrophic dataset error
- Inconsistent dataset allocation
- Device on which the dataset resides is down
- Inactive dataset entry in the system's Queued Dataset Table (QDT)

Format:

PDSDUMP,DN=*dn*,DV=*ldv*,{PDN | PDS}=*pdn*,ED=*ed*,CW=*cw*,ID=*uid*,

US=*usn*,OWN=*ov*,INC=*mm/dd/yy*:'*hh*:*mm*:*ss*',

ARC=*mm/dd/yy*:'*hh*:*mm*:*ss*',TS=*opt*,X,C,D,B,SO,I,O,S.

All parameters are in keyword form. Optional parameters identify which datasets are to be dumped or not dumped.

DN=*dn* — Name of dataset to which dump is written. The default is $PDS. Multiple dumps to a dataset are possible; if the dataset specified already exists, the dump is appended to it.

DV=*ldv* — Dumps all datasets residing on logical device *ldv*. Currently only one *ldv* can be specified.† Datasets can be limited by the B parameter.

PDN=*pdn* or PDS=*pds* — Dumps all editions of the specified permanent dataset. Editions can be limited by ED parameter.†

ED=*ed* — Edition number of permanent dataset dumped; meaningful only if PDS parameter is specified.†

† By default, all permanent datasets that could be specified by the parameters are dumped.

CW=*cw* Installation-defined control word regulating use of PDSDUMP. If the CW parameter is omitted, only the datasets belonging to the job owner can be dumped. If the CW parameter is present and the correct control word is used, any dataset can be dumped. If an invalid control word is given, the job step is aborted.

ID=*uid* Dumps all datasets with additional user identification as specified.† If ID is specified without a value, all datasets which meet the rest of the criteria and have a null ID are dumped.

US=*usn* Dumps all datasets with specified user number.†

OWN=*ov* Dumps all datasets with specified ownership value.†

INC=*mm/dd/yy*:'*hh*:*mm*:*ss*'
Incremental dump. Dumps only datasets modified since the specified date and time.

ARC=*mm/dd/yy*:'*hh*:*mm*:*ss*'
Archive datasets. Dumps and deletes datasets, regardless of the D option, that have not been accessed since the specified date and time.

TS=*opt* Timestamp conversion option. *opt* may be:

NS	Writes timestamp in nanosecond (new) format.
RT	Writes timestamp in real-time clock (old) format.
SAME	Does not convert timestamp.
CURR	Writes timestamp in whatever format is the current system default for writing timestamps.

If TS is not specified, TS=CURR is assumed.

X Dumps expired datasets

C Dumps selected datasets never dumped or datasets modified or adjusted since the last dump of the dataset

D Deletes datasets that are dumped

B Dumps only datasets that begin on the logical device specified by the DV parameter

SO Performs selection only (suppress actual dumping or deleting)

---

† By default, all permanent datasets that match the criteria specified by the parameters are dumped.

I	Dumps system input datasets	See the following note.
O	Dumps system output datasets	
S	Dumps user permanent datasets	

NOTE

If none of these parameters is specified, the input, output, and user permanent datasets are all dumped. If any of these parameters is specified, only those datasets of the type specified are dumped.

Multiple calls to PDSDUMP can be made if the dump dataset is to include several permanent datasets requiring specification of different parameters.

Example:

```
PDSDUMP,DN=DUMPA,PDS=LIB1.
PDSDUMP,DN=DUMPA,PDS=LIB2.
```

This example results in a dataset DUMPA that contains all editions of LIB1 and all editions of LIB2.

PDSDUMP produces a listing (see figure 11-1) on $OUT identifying the datasets dumped or bypassed and summarizing the dump run. The date and time in the heading line refer to the time when the dump run started. The permanent dataset name, edition number, ID, and user number are extracted from the DSC entry for each dataset selected. Each message is followed by the notation DUMPED, DUMPED AND DELETED, or NOT DUMPED. The notation NOT DUMPED indicates the dataset was selected but could not be accessed for dumping. A user logfile message further explains the problem encountered.

When dumping to a tape dataset, the recording format for the tape dataset must be transparent (for example, DF=TR on ACCESS statement). If the dataset is recorded in interchange format, loading of the dumped datasets cannot be performed.

PDSLOAD - LOAD PERMANENT DATASETS

PDSLOAD loads permanent datasets from a dataset created by PDSDUMP. If any of the permanent datasets already exist on Cray mass storage, it is reloaded only if the RP parameter is present.

```
PDSDUMP - PERMANENT DATASET DUMP UTILITY   DUMP ON 01/07/82 AT   14:50:44
AUDPL           ED=0001 ID=QITTYQAT USR=SYSTEM          DUMPED
AUDPL           ED=0002 ID=QITTYQAT USR=SYSTEM          DUMPED
DSCED           ED=0001 ID=QITTYQAT USR=SYSTEM          DUMPED
DSCED           ED=0002 ID=QITTYQAT USR=SYSTEM          DUMPED
TXBUILD         ED=0001 ID=QITTYQAT USR=SYSTEM          DUMPED
TXBUILD         ED=0002 ID=QITTYQAT USR=SYSTEM          DUMPED
TXBUILD         ED=0003 ID=QITTYQAT USR=SYSTEM          DUMPED
LONGDATASETNAME ED=0001 ID=QITTYQAT USR=SYSTEM          DUMPED
LONGDATASETNAME ED=0002 ID=QITTYQAT USR=SYSTEM          DUMPED
LONGDATASETNAME ED=0003 ID=QITTYQAT USR=SYSTEM          DUMPED
LONGDATASETNAME ED=0004 ID=QITTYQAT USR=SYSTEM          DUMPED
DSBUILD         ED=0001 ID=QITTYQAT USR=SYSTEM          DUMPED
DSBUILD         ED=0002 ID=QITTYQAT USR=SYSTEM          DUMPED
DSBUILD         ED=0003 ID=QITTYQAT USR=SYSTEM          DUMPED
DSBUILD         ED=0004 ID=QITTYQAT USR=SYSTEM          DUMPED
AUDPL           ED=0003 ID=QITTYQAT USR=SYSTEM          DUMPED
DSCED           ED=0003 ID=QITTYQAT USR=SYSTEM          DUMPED
TXBUILD         ED=0004 ID=QITTYQAT USR=SYSTEM          DUMPED
AUDPL           ED=0004 ID=QITTYQAT USR=SYSTEM          DUMPED
DSCED           ED=0004 ID=QITTYQAT USR=SYSTEM          DUMPED
           20 DATASETS SELECTED FOR DUMPING
```

Figure 11-1. PDSDUMP listing

Format:

PDSLOAD,L=*ldn*,DN=*dn*,{PDN | PDS}=*pds*,ED=*ed*,CW=*cw*,ID=*uid*,NID=*nuid*,

US=*usn*,OWN=*ov*,NOWN=*nov*,DV=*dvn*,RP,CR,A,I,O,S,NA,SO,TLA.

All parameters are in keyword form. Optional parameters identify which datasets are to be loaded or not loaded.

L=*ldn* — List dataset name. The default is $OUT.

DN=*dn* — Name of dataset from which permanent datasets are to be loaded. The default is $PDS.

PDN=*pdn* or PDS=*pdn* — Loads all editions of the specified permanent dataset. Editions can be limited by the ED parameter.†

ED=*ed* — Edition number of dataset to be loaded; meaningful only if PDS parameter is specified.†

CW=*cw* — Installation-defined control word regulating the use of PDSLOAD. If CW is omitted, only datasets belonging to the job owner are loaded.

ID=*uid* — Loads all datasets with additional user identification as specified

NID=*nuid* — Loads selected datasets with new user identification. This parameter is used to change the user identification of selected datasets.

US=*usn* — Loads all datasets with specified user number†

OWN=*ov* — Loads all datasets with specified ownership value†

NOWN=*nov* — Loads selected datasets to owner *nov*. This parameter is used to change the ownership value of the selected datasets.

DV=*dvn* — Name of logical device where the output dataset is assigned before it is opened. If omitted, COS assigns a device at open time. If this parameter is specified, the supplied device name is requested for the output dataset (the one being loaded). Note that COS can choose not to honor this assignment (for example, the device might not be currently available). This parameter is not involved in any way in the selection of a dataset for loading.

RP — If any of the specified datasets already exists, replaces with the one being loaded.

CR — Loads the most current version of a dataset, based on creation time. This option allows incremental loads to be performed in any order.

† By default, all permanent datasets that could be specified by the parameters are loaded.

A Loads only active datasets; that is, does not load expired datasets.

I Loads input datasets

O Loads output datasets — See the following note.

S Loads saved datasets

NOTE

If none of these parameters is specified, the input, output, and saved datasets are loaded. If any of these parameters is specified, only those datasets of the type specified are loaded.

NA Does not abort if there is not a dataset matching the specifications to load on the $PDS dataset. This parameter applies only to this situation. It does not prevent any other abort condition from occurring or offer reprieve processing of any kind.

SO Performs selection only; suppresses actual loading of datasets.

TLA Updates time of last access as the time that the load was performed.

PDSLOAD produces a listing on the list dataset identifying the datasets loaded or bypassed and summarizing the load run (see figure 11-2). The date and time in the heading line refer to the time when the load run started. The permanent dataset name, edition number, ID, and user number are extracted from the PDD for each dataset selected and successfully loaded. Each message is followed by the notation LOADED or NOT LOADED. The notation NOT LOADED indicates the dataset was selected but not loaded. An error message further explains the problem encountered.

```
 PDSLOAD - PERMANENT DATASET RESTORE UTILITY  LOAD ON 01/07/82 AT 17:13:47
ENTIT          ED=0001 ID=TAQI      USR=SYSTEM           LOADED
DSBUILD        ED=0001 ID=TAQI      USR=SYSTEM           LOADED
TXBUILD        ED=0001 ID=TAQI      USR=SYSTEM           LOADED
AUDPL          ED=0001 ID=TAQI      USR=SYSTEM           LOADED
DSCED          ED=0001 ID=TAQI      USR=SYSTEM           LOADED
             5 DATASETS SELECTED FOR LOADING
```

Figure 11-2. PDSLOAD listing

AUDIT - AUDIT PERMANENT DATASETS

The AUDIT utility provides reports on the status of each permanent dataset known to the system. AUDIT does not include input and output datasets.

If more than one parameter is selected, only those datasets which meet all criteria are listed.

AUDIT supplies the following information on the listing:

Permanent dataset name	Creation date/time
Dataset identifier	Last dump date/time
Edition number	Last access date/time
User identifications	Last modification date/time
Dataset size in words	Device name
Retention time in decimal	*note* information
Number of accesses in decimal	*text* information
Public access mode	Permitted users
Total block count in decimal	Access counts by user
Track access flag setting	Number of datasets selected

Format:

```
AUDIT,L=ldn,B=bdn,PDN=pdn,ID=uid,US=usn,DV=dvn,SZ=dsz,
ACC=opt:opt,X=mm/dd/yy:'hh:mm:ss',TCR=mm/dd/yy:'hh:mm:ss',
TLA=mm/dd/yy:'hh:mm:ss',TLM=mm/dd/yy:'hh:mm:ss',CW=cw,
OWN=ov,LO=opt:opt:opt:opt:opt,BO=opt:opt:opt:opt.
```

Parameters are in keyword form.

L=*ldn*	List dataset name. The default is $OUT.
B=*bdn*	Name of dataset to receive the binary output. If B is specified alone, the dataset is $BINAUD. If the B parameter is omitted, no binary output is written. For a description of the binary output format, see the Binary Audit Table description in the COS Table Descriptions Internal Reference Manual, publication SM-0045.
PDN=*pdn*	Name of permanent dataset or datasets to be listed
ID=*uid*	List all permanent datasets with the specified additional user identification. The default is to list all IDs. If ID is present without an equated value, datasets having a null ID are selected.
US=*usn*	List all permanent datasets with the specified user number. The default is to list all user numbers.
DV=*dvn*	List all permanent datasets on the specified logical device. The default is to list permanent datasets on all devices.
SZ=*dsz*	List all permanent datasets greater than or equal to the specified size. Size is specified in words. The default is to list all sizes.

ACC=*opt*:*opt*

Access option parameters. The options are:

AM List only those datasets belonging to OWN which have an explicit permit for the job's ownership value.

PAM List only those datasets belonging to OWN which have any form of public access (R:W:M:E).

If the OWN parameter is omitted, all datasets are searched for the permit or public access. If the CW parameter is specified, the AM includes any permit for any owner value. If the OWN parameter is specified and the CW and ACC parameters are omitted, AUDIT assumes the ACC=AM:PAM parameter on the control statement.

X=*mm/dd/yy*:'*hh*:*mm*:*ss*'

List all permanent datasets expired as of the specified *mm/dd/yy*:'*hh*:*mm*:*ss*'. *mm/dd/yy* can be specified alone. The default expiration date and time are "now" if only X is specified.

TCR=*mm/dd/yy*:'*hh*:*mm*:*ss*'

List all permanent datasets that have been created since the specified *mm/dd/yy*:'*hh*:*mm*:*ss*'. The keyword cannot be specified alone; however, TCR=*mm/dd/yy* is sufficient.

TLA=*mm/dd/yy*:'*hh*:*mm*:*ss*'

List all permanent datasets that have not been accessed since the specified *mm/dd/yy*:'*hh*:*mm*:*ss*'. The keyword cannot be specified alone; however, TLA=*mm/dd/yy* is sufficient.

TLM=*mm/dd/yy*:'*hh*:*mm*:*ss*'

List all permanent datasets that have been modified since the specified *mm/dd/yy*:'*hh*:*mm*:*ss*'. The keyword cannot be specified alone; however, TLM=*mm/dd/yy* is sufficient.

CW=*cw* Installation-defined control word regulating use of AUDIT. If the CW parameter is omitted, only the datasets belonging to the job owner can be listed. If the CW parameter is present and the correct control word is used, any dataset can be listed. If an invalid control word is given, the job step is aborted. When the CW and ACC parameters are omitted, but the OWN parameter is specified, AUDIT assumes the ACC=AM:PAM parameter on the control statement.

OWN=*ov* List all permanent datasets with the specified ownership value. If OWN is not specified, the job's ownership value is used.

Output formatting parameters:

LO=*opt*:*opt*:*opt*:*opt*:*opt*
Listing option selection. The options are:

S Short list which includes PDN, ID, and ED listed two per line. This is the default for interactive jobs when LO is not specified. This list option cannot be mixed with any others.

The following options can be specified alone or in combination separated by colons:

L Long list which includes PDN, ID, ED, size in words, retention time, access count, track access flag, public access mode, creation, last access, last modification, last dump time, and device name. L is the default for batch jobs when LO is not specified.

P Permit list which includes permitted owner name, access mode, access count, time of last access, and time of permit creation

A Access tracking which includes accessing owner name, access count, time of last access, and time of first access

T Text list which displays the dataset catalog *text* field

N Notes list which displays the dataset catalog *notes* field

BO=*opt*:*opt*:*opt*:*opt*
Binary audit options. These options specify what additional information, if any, is to be added to the standard binary audit file. They are ignored without comment unless a binary audit is requested (via the B parameter). If more than one option is desired, separate them with colons. The options are:

P Permits; one permit record is generated for each permitted user for each selected dataset.

A Access tracking; one record is generated for each accessing user for each selected dataset.

T Text; one record is generated for each selected dataset that has *text*.

N Notes; one record is generated for each selected dataset that has *notes*.

Figures 11-3 through 11-7 illustrate some of the LO options as they appear when the listing is directed to a mass storage dataset. Interactive reports omit the page header line. Non-private systems suppress the owner line unless OWN is used as a control statement parameter.

```
AUDIT             COS 1.12           05/24/83  12:35:33          PAGE      1

--------------------------    OWN = UVWXY      -------------------------

PDN                ID           ED        PDN               ID          ED

$DEBUG             DJB           1        $DS               DJB          5
$OVL               DJB           5        ARCHIVE           DJB          1
ARCHIVE            DJB           2        AUDIT             DJB          1
COSNL              DJB           1        ISAMPL            DJB          1
PROFILE            DJB           1

       9 DATASETS,     3099 BLOCKS,            1585585 WORDS
```

Figure 11-3. AUDIT, LO=S listing

```
AUDIT              COS 1.12              05/24/83  12:35:45            PAGE       1

 OWN = UVWXY
 ID     = DJB

      PERMITTED USERS FOR PDN = ARCHIVE           ID = DJB

      USER               AM      ACC      LAST ACCESS          CREATED

      XYZ                RWM       0                      05/16/83 12:09:09
      ABCD               R         0                      05/20/83 06:46:13
      QRZX               RW        0                      05/20/83 06:46:28
      ZILCH              E         0                      05/20/83 06:46:49

      PERMITTED USERS FOR PDN = ARCHIVE           ID = DJB

      USER               AM      ACC      LAST ACCESS          CREATED

      XYZ                RWM       0                      05/16/83 12:09:09
      ABCD               R         0                      05/20/83 06:46:13
      QRZX               RW        0                      05/20/83 06:46:28
      ZILCH              E         0                      05/20/83 06:46:49


       9 DATASETS,      3099 BLOCKS,            1585585 WORDS
```

Figure 11-4. AUDIT, LO=P listing

```
AUDIT              COS 1.12            05/24/83  12:36:10            PAGE       1

 OWN = UVWXY
 ID     = DJB

----------------------------------------------------------------------------
PDN              ID           ED                LAST     LAST     LAST     DEVICE
        SZ     RT    ACC TA PAM   CREATED ACCESSED MODIFIED  DUMPED
----------------------------------------------------------------------------

$DEBUG           DJB           1  05/16/83 05/16/83          05/20/83  DD-A1-24
      5574    45       4 N  RWM  11:47:36 12:22:27          06:02:22

$DS              DJB           5  03/29/83 05/18/83 05/14/83 05/20/83  DD-A2-20
      4608    45       7 N  N    10:45:29 14:27:09 15:08:22 06:03:00

   NOTES:

   THE FOLLOWING NOTES LINE IS MORE THAN 72 CHARACTERS IN LENGTH.
   123456789012345678901234567890123456789012345678901234567890123456789012
   34567890
   THE NEXT LINE IS ONLY ONE CHARACTER LONG.
   1

----------------------------------------------------------------------------
PDN              ID           ED                LAST     LAST     LAST     DEVICE
        SZ     RT    ACC TA PAM   CREATED ACCESSED MODIFIED  DUMPED
----------------------------------------------------------------------------

$OVL             DJB           5  03/29/83 05/14/83          05/20/83  DD-A1-21
     39424    45       6 N  RWM  10:45:29 17:15:38          06:05:29

   NOTES:

   SAMPLE NOTES DXT

----------------------------------------------------------------------------
PDN              ID           ED                LAST     LAST     LAST     DEVICE
        SZ     RT    ACC TA PAM   CREATED ACCESSED MODIFIED  DUMPED
----------------------------------------------------------------------------

ARCHIVE          DJB           1  05/12/83 05/20/83          05/20/83  DD-A1-24
      4096    45       4 N  RWM  11:18:10 06:44:22          06:04:01

      PERMITTED USERS:
```

Figure 11-5. AUDIT, LO=L:P:N listing

```
AUDIT             COS 1.12            05/24/83  12:36:10            PAGE       2

 OWN = UVWXY

      USER              AM      ACC     LAST ACCESS          CREATED

      XYZ               RWM       0                      05/16/83 12:09:09
      ABCD              R         0                      05/20/83 06:46:13
      QRZX              RW        0                      05/20/83 06:46:28
      ZILCH             E         0                      05/20/83 06:46:49

--------------------------------------------------------------------------------
PDN            ID           ED              LAST     LAST      LAST     DEVICE
        SZ    RT    ACC TA PAM    CREATED ACCESSED MODIFIED   DUMPED
--------------------------------------------------------------------------------

ARCHIVE        DJB           2  05/20/83 05/20/83           05/20/83  DD-A2-21
      3671    45       1 N  RWM  06:45:12 06:45:12           17:08:48

      PERMITTED USERS:

      USER              AM      ACC     LAST ACCESS          CREATED

      XYZ               RWM       0                      05/16/83 12:09:09
      ABCD              R         0                      05/20/83 06:46:13
      QRZX              RW        0                      05/20/83 06:46:28
      ZILCH             E         0                      05/20/83 06:46:49

--------------------------------------------------------------------------------
PDN            ID           ED              LAST     LAST      LAST     DEVICE
        SZ    RT    ACC TA PAM    CREATED ACCESSED MODIFIED   DUMPED
--------------------------------------------------------------------------------

AUDIT          DJB           1  05/24/83 05/24/83                     DD-A1-22
     26467    45       3 N  RWM  10:13:33 12:35:30

COSNL          DJB           1  04/06/83 04/07/83           05/20/83  DD-A2-20
   1498112    45       3 N  RWM  11:28:00 09:41:58           06:05:04

ISAMPL         DJB           1  08/11/81 04/22/83 03/03/83 05/20/83  DD-A2-20
      3584   100      24 N  RWM  10:07:41 17:21:54 10:02:58 06:04:46

PROFILE        DJB           1  04/30/83 05/24/83           05/20/83  DD-A2-21
        49    45      52 N  RWM  14:10:28 10:13:32           06:02:54


       9 DATASETS,     3099 BLOCKS,            1585585 WORDS
```

Figure 11-5. AUDIT, LO=L:P:N listing (continued)

```
AUDIT              COS 1.12              05/24/83  12:35:37            PAGE      1

 OWN = UVWXY
 ID     = DJB

------------------------------------------------------------------------------
PDN             ID          ED               LAST     LAST     LAST     DEVICE
        SZ     RT    ACC TA PAM   CREATED ACCESSED MODIFIED  DUMPED
------------------------------------------------------------------------------


$DEBUG          DJB          1  05/16/83 05/16/83          05/20/83  DD-A1-24
      5574     45      4 N  RWM 11:47:36 12:22:27          06:02:22

$DS             DJB          5  03/29/83 05/18/83 05/14/83 05/20/83  DD-A2-20

      4608     45      7 N  N   10:45:29 14:27:09 15:08:22 06:03:00

$OVL            DJB          5  03/29/83 05/14/83          05/20/83  DD-A1-21
     39424     45      6 N  RWM 10:45:29 17:15:38          06:05:29

ARCHIVE         DJB          1  05/12/83 05/20/83          05/20/83  DD-A1-24
      4096     45      4 N  RWM 11:18:10 06:44:22          06:04:01

ARCHIVE         DJB          2  05/20/83 05/20/83          05/20/83  DD-A2-21
      3671     45      1 N  RWM 06:45:12 06:45:12          17:08:48

AUDIT           DJB          1  05/24/83 05/24/83                    DD-A1-22
     26467     45      3 N  RWM 10:13:33 12:35:30

COSNL           DJB          1  04/06/83 04/07/83          05/20/83  DD-A2-20
   1498112     45      3 N  RWM 11:28:00 09:41:58          06:05:04

ISAMPL          DJB          1  08/11/81 04/22/83 03/03/83 05/20/83  DD-A2-20
      3584    100     24 N  RWM 10:07:41 17:21:54 10:02:58 06:04:46

PROFILE         DJB          1  04/30/83 05/24/83          05/20/83  DD-A2-21
        49     45     52 N  RWM 14:10:28 10:13:32          06:02:54


       9 DATASETS,     3099 BLOCKS,            1585585 WORDS
```

Figure 11-6. AUDIT, LO=L listing

```
AUDIT               COS 1.12              05/24/83  12:35:53              PAGE        1

 OWN = UVWXY
 ID     = DJB

   NOTES FOR PDN = $DS                    ID = DJB        ED =     5

   THE FOLLOWING NOTES LINE IS MORE THAN 72 CHARACTERS IN LENGTH.
   123456789012345678901234567890123456789012345678901234567890123456789012
   34567890
   THE NEXT LINE IS ONLY ONE CHARACTER LONG.
   1

   NOTES FOR PDN = $OVL                   ID = DJB        ED =     5

   SAMPLE NOTES DXT


       9 DATASETS,      3099 BLOCKS,            1585585 WORDS
```

Figure 11-7. AUDIT, LO=N listing (AUDIT, LO=T is nearly identical)

LOCAL DATASET UTILITIES 12

Local dataset utilities provide the user with a convenient means of copying, positioning, or initializing local datasets. The following utilities are available to the user:

- COPYR, COPYF, and COPYD copy blocked records, files, and datasets, respectively.
- COPYU copies unblocked datasets.
- SKIPR, SKIPF, and SKIPD skip blocked records, files, and datasets, respectively.
- SKIPU skips sectors on unblocked datasets.
- REWIND positions a blocked or unblocked dataset at the beginning of data, that is, before the first word of the dataset.
- WRITEDS initializes a blocked random or sequential dataset.

COPYR - COPY BLOCKED RECORDS

The COPYR utility copies a specified number of records from one blocked dataset to another starting at the current dataset position. Following the copy, the datasets are positioned after the EOR for the last record copied. The COPYR control statement is described below.

Format:

COPYR,I=*idn*,O=*odn*,NR=*n*.

Parameters are in keyword form.

I=*idn* Name of dataset to be copied. The default is $IN.

O=*odn* Name of dataset to receive the copy. The default is $OUT.

NR=*n* Decimal number of records to copy. The default is 1. If the dataset contains fewer than *n* records, the copy terminates on the next EOF. EOF or EOD is not written. If the keyword NR is specified without a value, the copy terminates at the next EOF. If the input dataset is positioned midrecord, the partial record is counted as one record.

COPYF - COPY BLOCKED FILES

The COPYF utility copies a specified number of files from one blocked dataset to another starting at the current dataset position. Following the copy, the datasets are positioned after the EOF for the last file copied. The COPYF control statement is described below.

Format:

```
COPYF,I=idn,O=odn,NF=n.
```

Parameters are in keyword form.

I=*idn* Name of dataset to be copied. The default is $IN.

O=*odn* Name of dataset to receive the copy. The default is $OUT.

NF=*n* Decimal number of files to copy. The default is 1. If the dataset contains fewer than *n* files, the copy terminates on EOD. EOD is not written. If the keyword NF is specified without a value, the copy terminates at the EOD. If the input dataset is positioned midfile, the partial file counts as one file.

COPYD - COPY BLOCKED DATASET

The COPYD utility copies one blocked dataset to another starting at their current positions. Following the copy, both datasets are positioned after the EOF of the last file copied. The EOD is not written to the output dataset.

Format:

```
COPYD,I=idn,O=odn.
```

Parameters are in keyword form.

I=*idn* Name of dataset to be copied. The default is $IN.

O=*odn* Name of dataset to receive the copy. The default is $OUT.

COPYU - COPY UNBLOCKED DATASETS

The COPYU utility copies a specified number of sectors or all data until EOD. The copy is made to or from the current position on both datasets. At the end of the copy, the datasets remain positioned after the last sector copied. The COPYU control statement is described below.

Format:

```
COPYU,I=i,O=o,NS=ns.
```

Parameters are in keyword form.

I=*i* Name of unblocked dataset to be copied.

O=*o* Name of unblocked dataset to receive the copy.

NS=*ns* Decimal number of sectors to copy. The default is 1. If the unblocked dataset contains fewer than *ns* sectors, the copy terminates on EOD. If the keyword *ns* is specified without a value, the copy terminates at EOD also.

Parameters I and O are required, and no default is given for them.

SKIPR - SKIP BLOCKED RECORDS

The SKIPR utility directs the system to bypass a specified number of records from the current position of the named blocked dataset. The SKIPR control statement is described below.

Format:

```
SKIPR,DN=dn,NR=n.
```

Parameters are in keyword form.

DN=*dn* Name of dataset to be bypassed. The default is $IN.

NR=*n* Decimal number of records to skip. The default is 1. If the keyword NR is specified without a value, the system positions *dn* after the last EOR of the current file. If *n* is negative, SKIPR skips backward on *dn*. If *dn* is positioned in the middle of the record, the partial record skipped counts as one record.

SKIPR does not bypass an EOF or beginning-of-data. If an EOF or beginning-of-data is encountered before *n* records have been bypassed when skipping backward, the dataset is positioned after the EOF or beginning-of-data. When skipping forward, the dataset is positioned after the last EOR of the current file. This statement is available for use with online tapes except that a negative value cannot be used for NR.

SKIPF - SKIP BLOCKED FILES

The SKIPF utility directs the system to bypass a specified number of files from the current position of the named blocked dataset. The SKIPF control statement is described below.

Format:

```
SKIPF,DN=dn,NF=n.
```

Parameters are in keyword form.

DN=*dn* Name of dataset to be bypassed. The default is $IN.

NF=*n* Decimal number of files to bypass. The default is 1. If the keyword NF is specified without a value, the system positions *dn* after the last EOF of the dataset. If *n* is negative, SKIPF skips backward on *dn*.

If *dn* is positioned midfile, the partial file skipped counts as one file.

SKIPF does not bypass an EOD or beginning-of-data. If beginning-of-data is encountered before *n* files have been bypassed when skipping backward, the dataset is positioned after the beginning-of-data. When skipping forward, the dataset is positioned before the EOD of the current file. This statement is available for use with online tapes except that a negative value cannot be used for NF; for interchange format tapes (DF=IC), NF can only be 1.

For example, if *dn* is positioned just after an EOF, the following control statement positions *dn* after the previous EOF. If *dn* is positioned midfile, *dn* will be positioned at the beginning of that file.

SKIPF,DN=*dn*,NF=-1.

SKIPD - SKIP BLOCKED DATASET

The SKIPD utility directs the system to position a blocked dataset at EOD, that is, after the last EOF of the dataset. It has the same effect as the following statement:

SKIPF,DN=*dn*,NF.

If the specified dataset is empty or already at EOD, the statement has no effect. The SKIPD control statement is described below.

Format:

SKIPD,DN=*dn*.

The parameter is in keyword form.

DN=*dn*	Name of dataset to be skipped. The default is $IN.

SKIPU - SKIP UNBLOCKED DATASET

The SKIPU utility directs the system to bypass a specified number of sectors or all data from the current position of the named unblocked dataset. The SKIPU control statement description follows.

Format:

```
SKIPU,DN=dn,NS=ns.
```

Parameters are in keyword form.

DN=*dn*	Name of unblocked dataset to be bypassed. There is no default value.

NS=*ns*	Decimal number of sectors to bypass. The default is 1. If the keyword NS is specified without a value, the system positions *dn* after the last sector of the dataset. If *ns* is negative, SKIPU skips backwards on *dn*.

REWIND - REWIND BLOCKED OR UNBLOCKED DATASET

The REWIND control statement positions the named datasets at the beginning-of-data. The $IN dataset represents an exception. After REWIND, $IN is positioned after the control statement file. REWIND opens any of the named datasets that are not open. REWIND is a system verb.

REWIND causes an EOD to be written to the dataset if the previous operation was a write or if the dataset is null. If the dataset is not memory resident, the buffers are flushed to mass storage when REWIND follows a write operation. If the dataset is memory resident, the EOD is still placed in the buffer, but the buffer is not flushed. For an online magnetic tape dataset, REWIND positions the tape dataset to the beginning of the first volume accessed by the user.

Format:

```
REWIND,DN=dn1:dn2:...:dn8.
```

Parameters are in keyword form.

DN=dn_i Names of datasets to be rewound. A maximum of eight datasets can be specified, separated by colons.

WRITEDS - INITIALIZE A BLOCKED RANDOM OR SEQUENTIAL DATASET

The WRITEDS utility is intended for initializing a blocked dataset. It writes a dataset containing a single file consisting of a specified number of records of a specified length. This utility is especially useful for random datasets because a record written on a random dataset must end on a pre-existing record boundary. Direct-access datasets, implemented in CFT as defined by the ANSI X3.9-1978 FORTRAN standard, can be initialized, and even extended, without the help of WRITEDS.

WRITEDS can also be used to write a sequential dataset.

The WRITEDS control statement is described below.

Format:

```
WRITEDS,DN=dn,NR=nr,RL=rl.
```

Parameters are in keyword form; the only required parameters are DN and NR.

DN=*dn* Name of dataset to be written. DN is a required parameter.

NR=*nr* Decimal number of records to be written. NR is a required parameter set to the largest value that may be needed, since a dataset is generally not extended when it is in random mode.

RL=*rl* Decimal record length, that is, the number of words in each record. The default is zero words, which generates a null record.

If the record length is 1 or greater, the first word of each record is the record number as a binary integer starting with 1.

ANALYTICAL AIDS 13

The following utilities provide analytical aids to the programmer:

- DUMPJOB and DUMP are generally used together to examine the contents of registers and memory as they were at a specific time during job processing. DUMPJOB captures the information so that DUMP can later format selected parts of it.
- DEBUG produces a symbolic dump.
- DSDUMP dumps all or part of a dataset to another dataset. The input dataset may be either blocked or unblocked.
- COMPARE compares two datasets and lists all differences.
- FLODUMP dumps flowtrace tables when a program aborts with flowtrace active.
- PRINT writes the value of an expression to the logfile.
- FTREF generates information about a FORTRAN application.
- SYSREF generates a global cross-reference listing for a group of CAL or APML programs.
- ITEMIZE inspects library datasets and generates statistics about them. Libraries are described in section 5; library dataset management is described in section 15.

DUMPJOB - CREATE $DUMP

The DUMPJOB control statement causes creation of the local dataset $DUMP, if not already existent. $DUMP receives an image of the memory assigned to the job (JTA and user field) when the DUMPJOB statement is encountered. Placing the DUMPJOB statement after a system verb, excluding the comment and EXIT statements, causes a dump of the Control Statement Processor (CSP). A DUMPJOB statement is not honored if an execute-only dataset is loaded in memory; a DUMPJOB to an execute-only dataset is rejected.

If the $DUMP dataset already exists, it is overwritten each time a DUMPJOB control statement is processed. If $DUMP is permanent and the job does not have write permission, DUMPJOB aborts. If $DUMP is permanent and the job has write permission, the dataset is overwritten.

If the DUMPJOB/DUMP sequence fails because of such situations as destroyed system-managed Dataset Parameter Areas, assign $DUMP before the job step for which the dump is to be written and save it with unique access. DUMPJOB writes to $DUMP, and job termination automatically adjusts $DUMP. $DUMP can then be inspected in a separate job.

$DUMP is created as an unblocked dataset by DUMPJOB for use by DUMP. DUMPJOB is a system verb and cannot be continued to subsequent statements.

Format:

```
DUMPJOB.
```

Parameters: None

DUMP - DUMP REGISTERS AND MEMORY

The DUMP utility reads and formats selected parts of the memory image contained in $DUMP and writes the information onto another dataset. The DUMP control statement can be placed anywhere in the control statement file after $DUMP has been created by the DUMPJOB control statement.

Placing the DUMPJOB and DUMP statements after an EXIT statement is conventional and provides the advantage of giving the dump regardless of which part of the job causes an error exit. The usage of DUMP and DUMPJOB, however, is not restricted to this purpose.

DUMP can be called any number of times within a job. This might be done to dump selected portions of memory from a single $DUMP dataset or it might be done if $DUMP has been created more than once in a single job.

Format:

DUMP,I=*idn*,O=*odn*,FWA=*fwa*,LWA=*lwa*,JTA,NXP,V,DSP,FORMAT=*f*,CENTER.

Parameters are in keyword form.

I=*idn* Name of the dataset containing the memory image. The dataset $DUMP is created by DUMPJOB and is the default, but any dataset in the $DUMP (unblocked) format is acceptable.

O=*odn* Name of the dataset to receive the dump; default is $OUT.

FWA=*fwa* First word address of memory to dump. The default is word 0 of JCB.

LWA=*lwa* Last word address of memory to dump. The default is word 200 of JCB. Specifying the keyword LWA without a value causes the limit address to be used. Specifying LWA=0 causes no memory to be dumped.

JTA Job Table Area to be dumped. The default is no JTA dump.

NXP No Exchange Package, B registers, T registers, cluster registers, or semaphore registers dumped. The default causes Exchange Package, B registers, T registers, cluster registers, and semaphore registers to be dumped. Cluster registers and semaphore registers are available only on CRAY X-MP mainframe types. NXP overrides the V parameter if the two are used together.

V Vector registers to be dumped. The default is no dump of V registers.

DSP Logical File Tables (LFTs) and Dataset Parameter Tables (DSPs) to be dumped. The default is to not dump LFTs and DSPs.

FORMAT=*f* Format for the part of memory selected by FWA and LWA. The first six of the following options are appropriate for formatting a dump of data. The I format is for a dump of of program instructions only.

- O Octal integer and ASCII character. This is the default.
- D Decimal integer and ASCII character
- X Hexadecimal integer and ASCII character
- G Floating-point or exponential, depending on the value of the number, and ASCII character
- P 16-bit parcel (4-word boundaries are forced for FWA and LWA)

M Mixed hexadecimal and octal written in ASCII. Each 16-bit parcel is represented as five characters; the first character is a hexadecimal digit representing the high-order 4 bits, and the next four are octal characters representing the low-order 12 bits.

I Instruction format. CAL instruction mnemonics printed with ASCII characters

CENTER Dump 100 (octal) words on each side of the address in the P register of the Exchange Package. The format is P.

Examples:

The following example is a portion of a data dump obtained using format O, the default format type:

```
*** Dump of memory from 00000000 to= 00000200 *** Format is O
*** Print Bias is:                    00010000 ***
******* Addresses from: 00000000 To: 00000077 Contain the same following pattern: 0000000000000000000000 *******
0000100 0405031464200000000000 0334000213400102175000 0000050215500002154200 0000213710500002154136 AC3D    7
0000104 0000010000375351600000 1144507000650140022000 0415172462013013430465 0000000000000000000000          (  A $ COS X.15
0000110 0000000000000000000000 0000000000000000000000 0000000000000000000000 0374400000000000000000                          ?
0000114 0000240000000000000000 0000000000000000655540 0000052712040000000002 0000000000040000141520                [   \
0000120 0000000000000001256241 0000000000000000000000 0000000000000000000000 0000000000000000000000        \
0000124 0000000000000000000000 0000000000000000000000 0000010000000000000000 0000000000000001771753
0000130 0000000005200000000124 0000077476540001771753 0000000000000000000000 0000000000000000000000    T    T
0000134 0000000000000000000000 0000000000000000000000 0000000000000000000000 0000000000000000000000
```

The same portion of the dump in format D:

```
*** Dump of memory from 00000000 to= 00000200 *** Format is D
*** Print Bias is:                    00010000 ***
******* Addresses from: 00000000 To: 00000077 Contain the same following pattern: 0000000000000000000000 *******
0000100     4702658803055722496    3963177258470406656      1417107280025728      4922006751991902 AC3D    7
0000104         281508995072000   -7410426839300037632   4850186722084794677                     0          (  A $ COS X.15
0000110                       0                      0                     0   4548635623644200960                          ?
0000114        5629499534213120                 220000      1509221443043330            4295017296                [   \
0000120                  351393                      0                     0                     0        \
0000124                       0                      0       281474976710656                521195
0000130            360777252948       2238515480359915                     0                     0    T    T
0000134                       0                      0                     0                     0
```

The same portion of the dump in format X:

```
*** Dump of memory from 00000000 to= 00000200 *** Format is X
*** Print Bias is:                    00010000 ***
******* Addresses from: 00000000 To: 00000077 Contain the same following pattern: 0000000000000000000000 *******
0000100       4143334400000000       370008B80108FA00       000508DA0008D880       00117C8A0008D85E AC3D    7
0000104       00010007EBA70000       9928E00D41802400       434F5320582E3135       0000000000000000          (  A $ COS X.15
0000110       0000000000000000       0000000000000000       0000000000000000       3F20000000000000                          ?
0000114       0014000000000000       0000000000035B60       00055CA100000002       000000010000C350                [   \
0000120       0000000000055CA1       0000000000000000       0000000000000000       0000000000000000        \
0000124       0000000000000000       0000000000000000       0001000000000000       000000000007F3EB
0000130       0000005400000054       0007F3EB0007F3EB       0000000000000000       0000000000000000    T    T
0000134       0000000000000000       0000000000000000       0000000000000000       0000000000000000
```

The same portion of the dump in format G:

```
*** Dump of memory from 00000000 to= 00000200 *** Format is G
*** Print Bias is:                   00010000 ***
******* Addresses from: 00000000 To: 00000077 Contain the same following pattern: 0000000000000000000000 *******
)000100      0.342195969964E+97     0.910133366849-695     0.000000000000E+00     0.000000000000E+00 AC3D    7              ^
)000104      0.000000000000E+00     0.000000000000E+00     0.304722870097+255     0.000000000000E+00           (  A $ COS X.15
)000110      0.000000000000E+00     0.000000000000E+00     0.000000000000E+00     0.000000000000E+00                         ?
)000114      0.000000000000E+00     0.000000000000E+00     0.000000000000E+00     0.000000000000E+00               [   \             P
)000120      0.000000000000E+00     0.000000000000E+00     0.000000000000E+00     0.000000000000E+00         \
)000124      0.000000000000E+00     0.000000000000E+00     0.000000000000E+00     0.000000000000E+00
)000130      0.000000000000E+00     0.000000000000E+00     0.000000000000E+00     0.000000000000E+00    T   T
)000134      0.000000000000E+00     0.000000000000E+00     0.000000000000E+00     0.000000000000E+00
```

The same portion of the dump in format P:

```
*** Dump of memory from 00000000 to= 00000200 *** Format is P
*** Print Bias is:                   00010000 ***
******* Addresses from: 00000000 To: 00000077 Contain the same following pattern: 0000000000000000000000 *******
0000100 040503 031504 000000 000000  033400 004270 000410 175000 AC3D    7
0000102 000005 004332 000010 154200  000021 076212 000010 154136              ^
0000104 000001 000007 165647 000000  114450 160015 040600 022000           (  A $
0000106 041517 051440 054056 030465  000000 000000 000000 000000 COS X.15
0000110 000000 000000 000000 000000  000000 000000 000000 000000
0000112 000000 000000 000000 000000  037440 000000 000000 000000        ?
0000114 000024 000000 000000 000000  000000 000000 000003 055540              [
0000116 000005 056241 000000 000002  000000 000001 000000 141520   \           P
0000120 000000 000000 000005 056241  000000 000000 000000 000000        \
```

The same portion of the dump in format M:

```
*** Dump of memory from 00000000 to= 00000200 *** Format is M
*** Print Bias is:                   00010000 ***
******* Addresses from: 00000000 To: 00000077 Contain the same following pattern: 0000000000000000000000 *******
0000100        4143334400000000      370008B80108FA00      0405031464200000000000 0334000213400102175000 AC3D    7
0000102        000508DA0008D880      00117C8A0008D85E      0000050215500002154200 0000213710500002154136               ^
0000104        00010007EBA70000      9928E00D41802400      0000010000375351600000 1144507000650140022000            (  A $
0000106        434F5320582E3135      0000000000000000      0415172462013013430465 0000000000000000000000 COS X.15
0000110        0000000000000000      0000000000000000      0000000000000000000000 0000000000000000000000
0000112        0000000000000000      3F20000000000000      0000000000000000000000 0374400000000000000000        ?
0000114        0014000000000000      0000000000035B60      0002400000000000000000 0000000000000000655540               [
0000116        00055CA100000002      000000010000C350      0000052712040000000002 0000000000040000141520   \            P
0000120        00000000000055CA1     0000000000000000      0000000000000001256241 0000000000000000000000       \
0000122        0000000000000000      0000000000000000      0000000000000000000000 0000000000000000000000
```

A dump of program instructions in format I:

```
******* Addresses from: 02154757 To: 02154776 Contain the same following pattern: 0000000000000000000000 *******
2154777a ERR       000                    ERR       000            0000000000000000000000
         ERR       000                    ERR       000
2155000a ERR       000                    ERR       000            0000000000000000040777          A
         ERR       000                    S7        17660554
2155001a                                  S5        S5>03          0605542667653026054530    al[}XXYX
         S1        S1<30                  S5        S5<30
2155002a S5        S5*FS1                 S0        S2\S5          0645512301243000200363    iiL
         JSZ       40074d
2155003a S1        15                                              0401000000641202101600    @@
         J         B10                    A6        4,A1
2155004a                                  A3        1,A1           0000044054000000231065          25
                                          A0        A6-A5
2155005a JAZ       40076d                                          0100010017546061225370        0 *
         A3         0+A5                  B70       A3
2155006a S0        1,A1                                            1210000000043400200406
         JSP       40101c
```

DEBUG - PRODUCE SYMBOLIC DUMP

The symbolic debug utility, DEBUG, provides a means of dumping portions of memory and interprets the dump in terms of FORTRAN or CAL symbols. DEBUG is normally used after an EXIT, DUMPJOB sequence when a job step aborts, but it can be used anywhere provided that a valid version of $DUMP exists.

In order for DEBUG to display variables, CFT and CAL must write special tables which the loader (LDR) augments with a version of the load map. The loader writes this information on a dataset called $DEBUG, which gives the FORTRAN or CAL symbol names associated with memory addresses. Table creation is initiated by specifying the ON=Z option for CFT or the SYM option for CAL. DEBUG reads $DEBUG and $DUMP and prints out variable names and values in a format appropriate for the variable type.

The following example shows the conventional use of DEBUG:

```
JOB, ... .
CFT,ON=Z.
LDR.
EXIT.
DUMPJOB.
DEBUG.
  .
  .
  .
```

Whether or not $DEBUG is present, DEBUG lists the status of job datasets and, for a multitasking program, the status of all existing tasks. Multitasking, stack, and heap statistics are reported whenever they are available.

The library routine SYMDEBUG is called from either FORTRAN or CAL with one argument, which is a Hollerith string containing any of the DEBUG parameters. SYMDEBUG produces output similar to that produced by DUMP but interprets the memory of the running program rather than $DUMP. It also uses the $DEBUG dataset. SYMDEBUG does not report the status of existing tasks or any statistics.

Neither DEBUG nor SYMDEBUG works with overlays or segmented loads.

The SYMS, NOTSYMS, BLOCKS, and NOTBLOCKS parameters permit a shorthand notation for the arguments specified. Using this notatation, a dash represents any number of characters or no characters and an asterisk represents any one character.

Examples:

SYMS=ABC-	Dump all symbols beginning with ABC.
SYMS=A***	Dump all 4-character symbols beginning with A.
SYMS=-A*-	Dump all symbols containing the letter A followed by one or more other characters.
SYMS=-	Dump all symbols.
SYMS=***-	Dump all symbols having three or more characters.

Format:

DEBUG,I=*idn*,O=*odn*,DUMP=*ddn*,TRACE=*n*,SYMS=*sym*,NOTSYMS=*nysm*,BLOCKS=*blk*,

NOTBLKS=*nblk*,MAXDIM=*dim*,TASKS,PAGES=*np*,COMMENTS='*string*'.

Parameters are in keyword form.

I=*idn* Name of dataset containing debug symbol tables. The default is $DEBUG, which is created by the loader from the symbol tables produced by CFT and CAL.

O=*odn* Name of dataset to receive the listing output from the symbolic debug routine. The default is $OUT.

DUMP=*ddn* Name of dataset containing the dump of the user field. This dataset is created by the DUMPJOB control statement. *ddn* is used when the symbolic debug routine is invoked after an abort. The default is $DUMP.

TRACE=*n* Number of routine levels to be looked at in symbolic dump. DEBUG traces back through the active subprograms the number of levels specified by *n*. If this parameter is omitted or if TRACE is specified without a value, the default is 50.

SYMS=*sym* List of symbols to be dumped by DEBUG. Up to 20 symbols can be specified; symbols are separated by a colon. This parameter applies to all blocks dumped. The default is all symbols.

NOTSYMS=*nsym*

List of symbols to be skipped. Up to 20 symbols can be specified; symbols are separated by a colon. This parameter applies to all blocks dumped. The default is that no symbols are to be skipped. This parameter takes precedence over the SYMS parameter.

BLOCKS=*blk*

List of common blocks to be included in the symbolic dump. A maximum of 20 blocks can be specified. All symbols (qualified by the SYMS and NOTSYMS parameters) in the blocks named here are to be dumped. If BLOCKS is specified without a value, all common blocks are dumped.

NOTBLKS=*nblk*

List of common blocks to be excluded from the symbolic dump. A maximum of 20 blocks can be specified. The default is to exclude no blocks. NOTBLKS specified without a value excludes all but the subprogram block. This parameter takes precedence over the BLOCKS parameter.

MAXDIM=*dim*

Maximum number of each dimension of the arrays to be dumped. This parameter allows the user to sample the contents of arrays without creating huge amounts of output. For example:

```
... ,MAXDIM=3:2:3, ...
```

causes the following elements to be dumped from an array dimensioned as A(10,3,6):

```
A(1, 1, 1)  A(2, 1, 1)  A(3, 1, 1)  A(1, 2, 1)  A(2, 2, 1)
A(3, 2, 1)  A(1, 1, 2)  A(2, 1, 2)  A(3, 1, 2)  A(1, 2, 2)
A(2, 2, 2)  A(3, 2, 2)  A(1, 1, 3)  A(2, 1, 3)
A(3, 1, 3)  A(1, 2, 3)  A(2, 2, 3)  A(3, 2, 3)
```

This parameter applies to all blocks dumped. The default is MAXDIM=20:5:2:1:1:1:1. The arrays are dumped in storage order.

TASKS — Trace back through all existing tasks; the default is to trace back only through tasks that were running when the dump dataset was written. Not available with SYMDEBUG.

PAGES=*np* — Page limit for the symbolic debug routine. The default is 70 pages.

COMMENT='*string*'

Identifier to be printed on the DEBUG output title line. Up to 8 ASCII characters can be specified.

DSDUMP - DUMP DATASET

The DSDUMP utility dumps specified portions of a dataset to another dataset. A disk dataset can be dumped in either blocked or unblocked format. A tape dataset can be dumped only in blocked format.

In the blocked format, a group of words within a record, a group of records within a file, and a group of files within a dataset can be selected. Initial word number, initial record number, and initial file number are relative to the current dataset position. Specifying an initial number greater than 1 (or 0, if the control statement includes the Z parameter) causes words, records, or files to be skipped starting from the current position.

Since the initial word, record, or file number is relative to the current position of the dataset, the dataset must be positioned properly before calling DSDUMP. A rewind of the dataset before calling DSDUMP makes the initial word, record, and file numbers relative to the beginning of the dataset. When DSDUMP is completed, the input dataset is positioned after the last record dumped.

The unblocked format is used for dumping a disk dataset without regard to whether it is blocked. Dumping a blocked dataset in unblocked format (by sectors) is possible. A group of sectors within the dataset or a group of words within each sector can be selected. The initial word and initial sector numbers are relative to the beginning of the dataset. Specifying an initial sector greater than 1 causes sectors to be skipped from the beginning of the dataset; specifying an initial word greater than 1 (or 0, if the control statement includes the Z parameter) causes words to be skipped from the beginning of each sector. Following a dump in unblocked format, the dataset is closed.

Two groups of DSDUMP parameters require the specification of numbers: the values of the initial word, record, file, and sector (I values) and their counts (N values). These values may be specified in three ways:

- Simple number (for example, 1234). This is interpreted as a decimal number.
- Explicit decimal number (for example, D'1234' or D1234).
- Explicit octal number (for example, O'1234' or O1234).

The following lines reference the same first word:

```
DSDUMP,...,IW=4096.
DSDUMP,...,IW=D'4096'.
DSDUMP,...,IW=O'10000'.
```

Format:

```
DSDUMP,I=idn,O=odn,DF=df,IW=n,NW=n,IR=n,NR=n,IF=n,
    NF=n,IS=n,NS=n,Z,DB=db,DSZ=sz.
```

Parameters are in keyword form; the only required parameter is I.

I=*idn* (or DN=*idn*)
: Name of dataset to be dumped. This is a required parameter.

O=*odn* (or L=*odn*)
: Name of dataset to receive the dump. The default is $OUT.

DF=*df*
: Dump format. The default is B.

  B Blocked
  U Unblocked

IW=*n*
: Decimal or octal number (*n*) of the initial word for each record/sector on *idn*. The default is 0 if Z is specified; 1 if Z is not included.

NW=*n*
: Decimal or octal number (*n*) of the words per record/sector to dump. Specifying NW without a value dumps all words to the end of a record/sector. The default is 1.

IR=*n*
: Decimal or octal number (*n*) of the initial record for each file on *idn*. Applicable only if DF=B. The default is 0 if Z is specified; 1 if Z is not included.

NR=*n*
: Decimal or octal number (*n*) of the records per file to dump. Specifying NR without a value dumps all records to the end of the file. Applicable only if DF=B. The default is 1.

IF=*n*
: Decimal or octal number (*n*) of the initial file for dataset on *idn*. Applicable only if DF=B. The default is 0 if Z is specified; 1 if Z is not included.

NF=*n*
: Decimal or octal number (*n*) of the files on *idn* to dump. Specifying NF without a value dumps all files to the end of the dataset. Applicable only if DF=B. The default is 1.

IS=*n*	Decimal or octal number (*n*) of the initial sector on *idn*. Applicable only if DF=U. The default is 0 if Z is specified; 1 if Z is not included.
NS=*n*	Decimal or octal number (*n*) of the sectors to dump. Specifying NS without a value dumps all sectors to the end of the dataset. Applicable only if DF=U. The default is 1.

Z — Zero-based initial-value parameters (IW, IR, IF, and IS). If Z is specified, the default value for each "I" parameter is 0, and output referring to word, record, file, and sector numbers begins at 0. The following lines reference the same first word:

```
DSDUMP,...,IW=4096.
DSDUMP,...,Z,IW=4095.
```

If Z is not specified, all "I" parameters are 1 based.

The Z parameter does not affect the "number of" ("N") parameters.

DB=*db* — Numeric base in which to display the data words.

OCTAL or O	Octal (base 8)
HEX or X	Hexadecimal (base 16)

The default is OCTAL.

DSZ=*sz* — Size of the data items to dump.

WORD or W	CRAY-1 words (64 bits)
PARCEL or P	CRAY-1 parcels (16 bits)

The default is WORD.

For blocked format, each record from *idn* dumped to *odn* is preceded by a header specifying the file and record number in both octal and decimal. For unblocked format, each sector is preceded by a header specifying the sector number in both octal and decimal.

Table 13-1 summarizes the DSDUMP output records according to the specification of DB and DSZ parameters.

Table 13-1. DSDUMP output format

DB,DSZ	Word count	Number interpretation	ASCII interpretation
OCTAL,WORD	†	Four 22-digit octal numbers	One 32-character interpretation
HEX,WORD	†	Four 16-digit hexadecimal numbers	One 32-character interpretation
OCTAL,PARCEL	†	Sixteen 6-digit octal numbers	None (insufficient space)
HEX,PARCEL	†	Sixteen 4-digit hexadecimal numbers	One 32-character interpretation

† If the Z parameter is used, the word count is 0 based and octal. If the Z parameter is not used, the word count is 1 based and decimal.

A row of five asterisks indicates that one or more groups of four words have not been formatted because they are identical to the previous four. Only the first group is formatted. The number of words not formatted can be determined from the word counts of the formatted lines before and after the asterisks. The final group of four or less words is always formatted.

COMPARE - COMPARE DATASETS

The COMPARE utility compares two blocked datasets and lists all differences found. The output consists of a listing of the location of each discrepancy, the contents of the differing portions of the datasets, and a message indicating the number of discrepancies. See the CRAY-OS Message Manual, publication SR-0039.

Keyword parameters allow the user to specify the maximum number of errors and the amount of context to be listed.

If portions of two datasets are being compared, the portions must be copied to a separate dataset before comparison; COMPARE compares complete datasets only.

COMPARE rewinds both input datasets before and after the comparison.

Format:

```
COMPARE,A=adn,B=bdn,L=ldn,DF=df,ME=maxe,CP=cpn,
    CS=csn,CW=cw1:cw2,ABORT=ac.
```

Parameters are in keyword form; both A and B must be specified.

A=*adn* and B=*bdn*
: Input dataset names. If *adn=bdn*, an error message is issued and the job step is aborted. A and B are required parameters.

L=*ldn*
: Dataset name for list of discrepancies. *ldn* must be different from *adn* and *bdn*. The default is $OUT.

DF=*df*
: Input dataset format. The default is T. *df* is a 1-character alphabetic code as follows:

  B Binary. The input datasets are compared logically to verify they are identical. If they are not identical, the differing words are printed in octal and as ASCII characters. The location printed is a word count in decimal. The first word of each dataset is called word 1.

  T Text. The input datasets are compared to see if they are equivalent as text. For example, a blank-compressed record and its expansion are considered equivalent. If the two datasets are not equivalent, the differing records are printed as text. The location is printed as a record count in decimal. The first record of each dataset is called record 1.

ME=*maxe*
: Maximum number of differences printed. The default is 100.

CP=*cpn*
: Amount of context printed. *cpn* records to either side of a difference are printed. The CP parameter applies only if DF=T; if DF=B and CP are specified, an error message is generated. The default is 0.

CS=*csn*
: Amount of context scanned. *csn* records to either side of a discrepancy are scanned for a match. The CS parameter applies only if DF=T; if DF=B and CS are specified, an error message is generated. The default is 0.

If a match is found within the defined range, subsequent comparisons are made at the same interval. That is, if record 275 of dataset A is equivalent to record 277 of dataset B, the next comparison is between record 276 of dataset A and record 278 of dataset B.

NOTE

If identical records occur within *csn* records of each other, the pairing is ambiguous and COMPARE can match the wrong pair.

CW=*cw* or CW=cw_1:cw_2
: Compare width. If CW=*cw* is specified, columns 1 through *cw* are compared. If CW=cw_1:cw_2 is specified, columns cw_1 through cw_2 are compared. Specifying CW without a value is not permitted. The default is to compare columns 1 through 133, but this can be changed by installation option. The CW parameter applies only if DF=T; if DF=B and CW are specified, an error message is generated.

ABORT=*ac*
: If *ac* or more differences are found, the job step aborts. Specifying ABORT alone is equivalent to ABORT=1 and causes an abort if any differences are found. Specifying ABORT does not prevent the listing of up to *maxe* differences.

PRINT - WRITE VALUE OF EXPRESSION TO LOGFILE

The PRINT control statement writes the value of an expression on the logfile. The value of the expression is written in three different formats: as a decimal integer, as a 22-digit octal value, and as an ASCII string. PRINT is a system verb.

Format:

PRINT(*expression*)

Parameter:

expression
Any JCL expression (see section 16). This parameter is required.

Logfile format:

UT060 *decimal octal ASCII*

UT060	Message code indicating origin is PRINT statement
decimal	16-digit decimal representation of evaluated expression
octal	22-digit octal representation of evaluated expression
ASCII	8-character ASCII representation of evaluated expression

FLODUMP - FLOW TRACE RECOVERY DUMP

The FLODUMP utility recovers and dumps flow trace tables when a program aborts with flow tracing active. The flow trace tables are dumped in the FORTRAN flow trace format.

FLODUMP is invoked by specifying the F option on the CFT control statement and including the FLODUMP control statement in the COS control statement file. (Refer to the FORTRAN (CFT) Reference Manual, CRI publication SR-0009, for more information on the F option.)

Format:

```
FLODUMP.
```

Parameters: None

The following example illustrates the use of the FLODUMP control statement.

```
JOB, ... .
CFT,ON=F.
LDR.
EXIT.
DUMPJOB.
FLODUMP.
.
.
.
```

A flow trace summary is illustrated in figure 13-1; a flow trace recovery dump is shown in figure 13-2.

The examples in figures 13-1 and 13-2 show that the total time reported for the main program, ONF, is larger for the flow trace recovery dump (FLODUMP) than for the flow trace summary. The difference is that the time reported with FLODUMP includes the main program's execution time, the time required to abort the program, and the time required to recover the flow trace tables.

```
 FLOW TRACE --- SUMMARY
        ROUTINE            TIME          %    CALLED    AVERAGE T

  1  ONF            0.000053     5.42         1      0.000053
                                                                 CALLS SUB1
  2  SUB1           0.000323    32.80         9      0.000036 CALLED BY ONF
                                                                 CALLS SUB2
  3  SUB2           0.000322    32.75         9      0.000036 CALLED BY SUB1
                                                                 CALLS SUB3
  4  SUB3           0.000286    29.04         9      0.000032 CALLED BY SUB2
***      TOTAL      0.000985
***  OVERHEAD       0.000712


SUBROUTINE LINKAGE OVERHEAD SUMMARY              28 CALLS

               MINIMUM   MAXIMUM   AVERAGE   CYCLES   SECONDS          %
T REGISTERS          1         2       2.0      838  1.05E-05     1.0640
B REGISTERS          2         3       3.0      894  1.12E-05     1.1351
  ARGUMENTS          0         0       0.0        0  0.00E+00     0.0000
      TOTAL                                    1732  2.17E-05     2.1991
 MAXIMUM SUBROUTINE DEPTH =  4
```

Figure 13-1. Example of a flow trace summary

```
  FLOW TRACE RECOVERY DUMP --- RECOVER WITH ONFDMP   ACTIVE
 FLOW TRACE --- SUMMARY
       ROUTINE        TIME        %  CALLED   AVERAGE T

    1 ONFDMP      0.000328  26.04       1    0.000328
                                                      CALLS SUB1
    2 SUB1        0.000323  25.64       9    0.000036 CALLED BY ONFDMP
                                                      CALLS SUB2
    3 SUB2        0.000322  25.61       9    0.000036 CALLED BY SUB1
                                                      CALLS SUB3
    4 SUB3        0.000286  22.70       9    0.000032 CALLED BY SUB2
***       TOTAL   0.001259
***    OVERHEAD   0.000712


SUBROUTINE LINKAGE OVERHEAD SUMMARY                  28 CALLS

              MINIMUM   MAXIMUM   AVERAGE      CYCLES   SECONDS
T REGISTERS         1         2       2.0         838  1.05E-05    0.83
B REGISTERS         2         3       3.0         894  1.12E-05    0.88
  ARGUMENTS         0         0       0.0           0  0.00E+00    0.00
      TOTAL                                      1732  2.17E-05    1.71
 MAXIMUM SUBROUTINE DEPTH =  4
```

Figure 13-2. Example of a flow trace recovery dump

FTREF - GENERATE FORTRAN REFERENCE LISTING

FTREF is a tool that generates a listing containing several forms of information about a FORTRAN application. FTREF reports on the common block variables used in the subroutines within an application. FTREF provides tabular information that consists of entry names, calling routines, and called routines for each subroutine and displays this information as a static calling tree. If the user program is multitasked, FTREF reports whether a common variable or a subroutine is locked when it is referenced or redefined.

FTREF requires the output produced when ON=XS is specified in a previous CFT statement. The dataset to be processed by FTREF may contain any number of modules used by the application. The more application modules included in the dataset, the more complete will be the output of FTREF.

FTREF CONTROL STATEMENT

The FTREF control statement follows a CFT control statement that specified ON=XS.

Format:

```
FTREF,I=idn,L=ldn,TREE=op,CB=op,ROOT=root,END=end,
                    LEVEL=n,DIR=dir,NORDER.
```

Parameters:

I=*idn* Input dataset containing the cross reference table listing and the source program from CFT with ON=XS. This parameter is required.

L=*ldn* Dataset containing the output listing. The default is $OUT.

TREE=*op* Produces information about the routines called and the static calling tree for the program. The value '**LOOP**' indicates there is an apparent recursive call in the program. The options are:

- PART Reports entry names, external calls, other routines that call the routine, and common block names from the input dataset.
- FULL Reports the information that the PART option provides plus the static calling tree.

The default is PART.

CB=*op* Global common block cross references. The options are:

- PART Identifies the routine names using a common block.
- FULL Details the use of the variables of a common block in a routine.

The default is PART.

ROOT=*root* Produces a tree with the routine *root* as a root. ROOT can be used to get a subtree for the program. As a default, the routine not called by another routine is chosen. If there are more than one uncalled routines, the first routine by alphabetic order is chosen as the root.

END=*end* Produces a tree with the routine *end* as the end of the tree. The value '**STOP**' is printed whenever the routine is found, and that branch of the tree is terminated. By default, FTREF generates a tree containing all subroutines in the program.

LEVEL=*n* Produces a tree of *n* levels. The default is the entire program. If both LEVEL and END are specified, FTREF terminates a branch of the tree at whichever state is encountered first.

DIR=*dir* Selects modules to process or common blocks for FTREF to check to determine whether a variable is in a locked area. The dataset *dir* contains a set of directives that control the processing or check. If DIR is specified without a value, the directives are taken from the next file of $IN. The directives are explained in detail in the next subsection. The default is no directives to be read.

NORDER Lists the subroutines in input order instead of alphabetic order. The default is alphabetic order.

DIRECTIVES

A FTREF directive consists of a keyword and zero or more parameters. A blank, comma, or open parenthesis separates the keyword and the parameters. A period, closed parenthesis, or two consecutive blanks are the terminator. A caret at the end of the directive line indicates that the next line is a continuation of the current directive. Do not precede the caret by a blank; it must immediately follow the last non-blank character of the line. The caret also separates the parameters from the next line.

One line may contain only one directive. FTREF processes 80 characters per line and ignores the rest of the characters.

SUBSET directive

The SUBSET directive specifies the modules to be processed by FTREF. This directive prevents FTREF from processing modules not on the parameter list. SUBSET may appear any time and as often as necessary.

If no SUBSET directive appears, or only one SUBSET directive is specified and it has no parameters, FTREF processes all modules in the input file.

Format:

SUBSET,md_1,md_2,...md_n.

where *md* is the name of the module to be processed by FTREF.

CHKBLK directive

The CHKBLK directive specifies the common blocks whose variables FTREF is to check to determine whether a variable is in a locked area. The directive may appear any time and as often as necessary. If no CHKBLK directive appears, FTREF does not check for variables in a locked area. If there is only one CHKBLK directive and it is without a parameter, FTREF processes all the block variables in the input.

A locked area begins with a LOCKON call and ends with a LOCKOFF call with the same lock variable. If a common block variable is referenced or redefined in such an area, it is considered to be locked; otherwise, it is unlocked. Check to ensure that this is a safe condition. A locked area must be completely within a compilation module to be detected by FTREF. See the Multitasking User Guide, CRI publication SN-0222, for more information on using the locking feature.

Format:

CHKBLK,blk_1,blk_2,...blk_n.

where *blk* is the name of the common block whose variables FTREF is to check to determine whether a variable is in a locked area. Specify // as the name of a blank common block.

NOTE

FTREF searches for the presence of the LOCKON and LOCKOFF calls. It does not attempt a flow analysis, and it does not consider the effects of IF statements.

CHKMOD directive

The CHKMOD directive specifies the external calls whose routines FTREF is to check to determine whether an external call is in a locked area. The directive may appear any time and as often as necessary. If no CHKMOD directive appears, or if there is only one CHKMOD directive and it is without a parameter, FTREF does not check for external calls in a locked area. See the CHKBLK directive above for a definition of a locked area.

Format:

CHKMOD,mod_1,mod_2,...mod_n.

where *mod* is the name of the module whose routines FTREF is to check to determine whether then module is called from a locked area or not.

Example:

The following example uses the FTREF control statement.

```
JOB,...
ACCOUNT,...
CFT,ON=XS,L=XYZ.
FTREF,I=XYZ,CB=FULL,TREE=FULL,DIR.
          .
          .
/EOF
     CFT source
/EOF
CHKBLK
/EOF
          .
          .
```

If the example uses the following tree structure,

```
                           MAIN
                             |
          ---------------------------------------
          |                  |                  |
         SUB1               SUB2               SUB3
          |                  |
   --------------            |
   |            |            |
SUBSUB1      SUBSUB2      SUBSUB1
   |                         |
SUB22                      SUB22
   |                         |
SUBSUB2                   SUBSUB2
```

. . . the portion of the output produced by TREE=FULL is:

```
                                    *******************************************
                                    *                                         *
                                    *     STATIC CALLING TREE FOR MAIN        *
                                    *                                         *
                                    *******************************************

MAIN      SUB1      SUBSUB1  SUB22    SUBSUB2
             2 >    SUBSUB2
   1 >    SUB2      SUBSUB1  SUB22    SUBSUB2
   1 >    SUB3
                                           ****** END OF THE CALLING TREE ******
```

SYSREF - GENERATE GLOBAL CROSS-REFERENCE LISTING

The SYSREF utility generates a global cross-reference listing for a group of CAL or APML programs. The number of CAL or APML programs that can be included in such a group is limited by the amount of Cray Computer System memory allocated to a user.

SYSREF reads special binary symbol tables written by CAL or APML and produces a single cross-reference listing for the program modules represented in the tables. When the X parameter appears on a CAL or APML statement, a record is written for each program unit assembled. The records are written to a dataset specified by the X parameter ($XRF by default or if X appears alone). Each record has a header containing the name of the program unit. The rest of the record consists of cross-reference information for every global symbol used in that program.

Format:

SYSREF,X=*xdn*,L=*ldn*.

Parameters:

X=*xdn* Name of dataset whose first file (normally the only file) contains one or more symbol records written by CAL and/or APML. The default is $XRF.

L=*ldn* Name of output dataset. The default is $OUT.

USE OF SYSREF

SYSREF is usually used to process symbol records written by CAL and/or APML earlier in the same job. To do so, add X parameters to each CAL or APML control statement and follow them with a SYSREF control statement:

```
CAL,X.
APML,X.
CAL,X.
SYSREF,L=XROUT.
```

$XRF is used as default in all cases.

To process symbol records written in an earlier job, the following sequence is used:

The first job:

```
CAL,X.
APML,X.
SAVE,DN=$XRF,ID=XX.
```

The second job:

```
ACCESS,DN=$XRF,ID=XX.
SYSREF,L=XROUT.
```

To add more symbol records before invoking SYSREF, use:

```
ACCESS,DN=$XRF,ID=XX,UQ.
SKIPR,DN=$XRF,NR.
CAL,X.
SYSREF.
```

The format above has the same effect as if the CAL step had been done before the SAVE step.

GLOBAL CROSS-REFERENCE LISTING FORMAT

The global cross-reference listing contains only global symbols. A symbol is global if it is any one of the following:

- Named in an ENTRY or EXTERNAL statement
- Defined before an IDENT statement and after any preceding END statement

- Defined within a system text such as $SYSTXT
- Defined within a section of source code bracketed by TEXT and ENDTEXT pseudo instructions

The order of the symbols in the global cross-reference listing is lexicographic, based first on the symbol name and then (within each symbol name) on the module name. An exception to the order is made for symbol names beginning with N@, S@, or W@. These symbol names are sorted as if @ is the most significant (leftmost) character and the N, S, or W is the least significant character. The listing displays the symbol name correctly. The effect is a grouping of all the N@, S@, and W@ symbols that refer to the same field in a table.

The global cross-reference listing consists of 13 columns:

Column	Heading	Contents
1	Value	The symbol's value
2	Symbol	The symbol's name
3	Origin	The IDENT of the system text in which the symbol is defined; or the label of the TEXT block in which the symbol is defined; or *GLOBAL*, if the symbol is defined outside any program unit; or blank.
4	Module	The IDENT of the module within or before which the symbol is defined or referenced
5-13	References	A list of the lines on which the symbol is defined or referenced

The symbol's name, value, and references appear in the same format as in a CAL or APML listing. The page number in each reference is a local page number which starts at 1 for each module. In a CAL or APML listing, this is the page number that appears in parentheses to the right of the second title line on each page.

ITEMIZE - INSPECT LIBRARY DATASETS

The ITEMIZE utility prints a formatted report of the contents of a dataset generated by CAL, CFT, BUILD, LDR, UPDATE, and other compatible processors. For additional information about the contents of an UPDATE PL, use AUDPL. See CRI publication SR-0013, UPDATE Reference Manual.

ITEMIZE is executed using the following control statement.

Format:

```
ITEMIZE,DN=dn,L=odn,NREW,NF=n,T,BL,E,B,X.
```

Parameters:

DN=*dn* — Local dataset name of the dataset to be listed. The default is $OBL.

L=*odn* — Local dataset name where listing is written. If L is omitted or is specified alone, $OUT is used.

NREW — No rewind. Specifies the dataset is not rewound. If NREW is omitted, the dataset to be listed is rewound before and after ITEMIZE is executed.

NF=*n* — Number of files within a dataset to be listed. If NF is used alone, the contents of all files within the dataset are listed. If NF=*n*, the contents of *n* files within the dataset are listed. The default is NF=1.

T — Truncation. Specifying this parameter truncates lines on the listing dataset to 80 characters. Optional parameter; however, specifying this parameter precludes specifying the E, B, and X parameters.

BL — Burstable listing. When this parameter is specified, each dataset heading starts at the top of a page. The default is a compact listing in which a page eject occurs only when the current page is nearly full.

E — Entry points. Specifying E causes all entry points to be included in the listing. Use for binary library datasets only.

B — Blocks. Specifying B causes all entry points, code, and common block information to be included in the listing. Use for binary library datasets only. (B overrides E.)

X — Externals. Specifying X causes all entry points, code, common block, and external information to be included in the listing. (X overrides B.)

Restrictions:

- An UPDATE PL is recognized only if it is the only item in a dataset.
- ITEMIZE operates on standard COS blocked datasets only.
- ITEMIZE does not operate on a tape dataset.

A header containing the jobname, ITEMIZE version number, date, time, and page number appears at the top of every page. The line shown below appears following the header on page 1 (or only page). The line gives the local dataset name of the dataset being processed.

ITEMIZE OF *dn*

ITEMIZE normally produces file-level output. However, for binary library datasets, it produces a more detailed record-level output. The following subsections describe both levels of output.

FILE-LEVEL OUTPUT

ITEMIZE prints one line for each file examined (up to the maximum specified by the NF parameter or the default of 1). A second header line appears on each page and contains the column headings shown in figure 13-3.

```
  TITEMA       ITEMIZE 1.08        05/10/82          08:58:15    PAGE   1
             ITEMIZE OF TESTPL
   FILE      RECORDS     TYPE      LENGTH    CHECK      PART      DATE
     1            6       PL           18     0650      0650    05/10/82
     2            5       PL           15     0512      0512    05/10/82
     3            4       PL           12     0313      0313    05/10/82
     4            1       PL            6     3075      3075    05/10/82
     5            1       PL            6     5756      5756    05/10/82
     0     * EOD *                     57     2334      2334
```

Figure 13-3. Sample listing of ITEMIZE for a PL

Figure 13-3 is an example of ITEMIZE operating on a program library. The control statement used to generate the listing was ITEMIZE,BL,NF.

FILE	Sequence number of the file within the dataset
RECORDS	Number of records within the file
TYPE	Type of information contained within the file. If the file is a member of a PL, the column contains PL. Other values which may appear in this column are ABS, REL, DAT, and ???. ABS and REL indicate absolute and relocatable program modules, respectively. DAT indicates data, and ??? is used for otherwise unrecognized files.
LENGTH	Length of the file in words
CHECK	Checksum of the data within the file
PART	This field is the same as CHECK for file-level output.
DATE	Date of the PL from its directory or blank if other types of datasets

A PL created by the UPDATE utility consists of many files. The last file of the dataset must be a PL directory. If NF is not specified on the control statement, ITEMIZE prints information only for the first files, although it has examined the last file. The dataset must contain only a PL.

OUTPUT FOR BINARY LIBRARY DATASETS

A binary library is a collection of binary records recognized by the existence of a Program Description Table (PDT) Table. For binary library datasets, ITEMIZE operates record-by-record rather than file-by-file. The second header line for binary library datasets contains the column headings shown in the following figure.

Figure 13-4 is an example of ITEMIZE operating on a binary library dataset. The control statement used to generate the listing was ITEMIZE,BL,NF,X. If the control statement had been ITEMIZE,BL,NF., lines with no entry in the REC column would not have appeared.

REC	Sequence number of the record within the file
NAME	Name of the program from the PDT
TYPE	ABS or REL. ABS and REL indicate absolute and relocatable program modules, respectively.

```
TITEMA      ITEMIZE 1.08        05/10/82        08:58:15     PAGE   1
        ITEMIZE OF TESTLIB                       FILE     1
  REC     NAME       TYPE        LENGTH    CHECK     PART      DATE
   1    DUMMY1        REL            41     6200     0344    05/10/82 08:58:14    CFT 1.09 03/25/82    COS 1.11 05/09/82
                                                COMMENT:
         * ENT *               DUMMY1
         * BLK *   DUMMY1        MODULE LENGTH :        11
         * BLK *   #TB           MODULE LENGTH :         4
         * EXT *               DUMMY2             DUMMY3
   2    DUMMY2        REL            38     2177     0244    05/10/82 08:58:14    CFT 1.09 03/25/82    COS 1.11 05/09/82
                                                COMMENT:
         * ENT *               DUMMY2
         * BLK *   DUMMY2        MODULE LENGTH :        10
         * BLK *   #TB           MODULE LENGTH :         4
         * EXT *               DUMMY3
   3    DUMMY3        REL            34     6403     0637    05/10/82 08:58:14    CFT 1.09 03/25/82    COS 1.11 05/09/82
                                                COMMENT:
         * ENT *               DUMMY3
         * BLK *   DUMMY3        MODULE LENGTH :         9
         * BLK *   #TB           MODULE LENGTH :         4
   1    * EOF *                     113     0742     0065

TITEMA      ITEMIZE 1.08        05/10/82        08:58:15     PAGE   2
        ITEMIZE OF TESTLIB                       FILE     2
  REC     NAME       TYPE        LENGTH    CHECK     PART      DATE
   1    * DIR *       REL            19     3512     3512
    DIRECTORY ID : D01          DIRECTORY LENGTH :         19 WORDS.
 MODULE NAME : DUMMY1  .  NO. OF BLOCKS    :     1,  NO. OF ENTRIES    :     1,  NO. OF EXTERNALS :     2
         * ENT *               DUMMY1
         * BLK*    #TB
         * EXT *               DUMMY2             DUMMY3
 MODULE NAME : DUMMY2  .  NO. OF BLOCKS    :     1,  NO. OF ENTRIES    :     1,  NO. OF EXTERNALS :     1
         * ENT *               DUMMY2
         * BLK*    #TB
         * EXT *               DUMMY3
 MODULE NAME : DUMMY3  .  NO. OF BLOCKS    :     1,  NO. OF ENTRIES    :     1,  NO. OF EXTERNALS :     0
         * ENT *               DUMMY3
         * BLK*    #TB
   2    * EOF *                      19     3512     3512
   0    * EOD *                     132     1130     0246
```

Figure 13-4. Sample listing of ITEMIZE for a binary library dataset with X and NF parameters

LENGTH	Length of the record in words
CHECK	Checksums
PART	Checksums
DATE	Date of compilation from the PDT

One line containing the data listed above is generated for each record. If any of the E, B, or X options are specified on the control statement, several additional lines can be printed. The information in these lines is labeled separately as described in the following paragraphs.

When E, B, or X is specified, the comment field of the PDT is printed on a separate line. In addition, the entry point names are printed with five names per line.

When B or X is specified, a separate line is printed for each block containing its name and length.

When X is specified, the externals referenced by the program are printed with five external names per line.

A binary library dataset contains a second directory file containing one record. If E, B, or X is specified on the control statement, a line is printed specifing the directory ID and length. In addition, entries, blocks, and externals are printed as described above for program records.

EXECUTABLE PROGRAM CREATION 14

The COS Relocatable Loader is a utility program that executes within the user field and provides the loading and linking in memory of relocatable modules from datasets on mass storage.

The relocatable loader is called through the LDR control statement when a user requires loading of a program in relocatable format. Absolute load modules can also be loaded. The design of the COS loader tables and relocatable loader allows program modules to be loaded, relocated, and linked to externals in a single pass over the dataset being loaded. This minimizes the time spent in loading activities on the Cray Computer System. The loader allows the immediate execution of the object module or the creation of an absolute binary image of the object module on a specified dataset. Loader features are governed by parameters of the LDR control statement.

The relocatable loader can also generate a partially relocated module. This module, referred to as a relocatable overlay, is described later in this section.

LDR CONTROL STATEMENT

The loader is called into execution by the LDR control statement. Parameters of the control statement determine the functions to be performed by the loader.

Format:

```
LDR,DN=dn,LIB=ldn,NOLIB=ldn,LLD,AB=adn,MAP=op,SID[='string'],T=tra,
NX,DEB=l,C=com,OVL=dir,CNS,NA,USA,L=ldn,SET=val,E=n,I=sdir,
NOECHO,SECURE,GRANT=sc1:sc2:...:scn,BC=bc,PAD=pad,NORED,
STK[=initial size[:increment]],MM[=initial size[:increment]],
MMEPS=epsilon,MMLOC={AFTER|BEFORE}.
```

Parameters are in keyword form.

DN=*dn* Dataset containing modules to be loaded. The default is $BLD. Loading continues until an end-of-file is reached. Modules are loaded according to block name as determined by a CAL IDENT statement or a CFT PROGRAM, SUBROUTINE, BLOCK DATA, or FUNCTION statement. Duplicate blocks are skipped and an informative message is issued.

Multiple files from the same dataset can be loaded by specifying the dataset name multiple times separated by colons. A maximum of eight files can be indicated.

Datasets specified by the DN parameter are closed at the end of the load process. Closing a dataset has the effect of rewinding the dataset and releasing I/O tables and buffers.

Modules to be loaded can be relocatable or absolute. However, the two types of modules cannot be mixed.

For example,

```
DN=LOAD1:LOAD2:$BLD
```

causes the loading of all modules in the first file of datasets LOAD1, then LOAD2, and then $BLD.

Normally the dataset is rewound before loading; however, consecutive occurrences of a dataset name inhibit subsequent rewind operations. Therefore, the statement

```
DN=LOAD3:LOAD3
```

causes the loading of all modules in the first two files of dataset LOAD3.

The DN parameter takes on a special quality when OVL is specified: only one *dn* can be specified. The dataset named is the initial LOAD file used by the overlay loader. (See the description of overlay loading later in this section for more information.)

LIB=*ldn* The LIB parameter names the dataset from which unsatisfied externals are loaded. A maximum of eight datasets can be named, with the dataset names separated by colons.

Any default libraries are automatically included in the library list unless the NOLIB parameter is specified. The loader accesses the default libraries from the COS System Directory (SDR) if they are not local to the job; no ACCESS statement is required.

Datasets specified by the LIB parameter are closed at the end of the load process. Closing a dataset has the effect of rewinding the dataset and releasing I/O tables and buffers.

NOTE

These datasets should be generated using the BUILD utility to prevent unnecessary overhead in the loader.

The libraries cannot be tape datasets.

NOLIB=*ldn* The NOLIB parameter value names the specific default library to be excluded from the load. Selecting NOLIB with no value specifies the exclusion of all default system libraries. If NOLIB is not specified, any default libraries that a site has are automatically included in the library list, along with any libraries specified on the LIB parameter.

LLD Specifying the LLD parameter causes any libraries included in the load to be retained as local datasets at load completion. These local datasets remain open. Datasets automatically accessed are not released at load completion. If the LLD parameter is not specified, the loader closes all libraries and releases automatically-accessed datasets at load completion.

AB=*adn* Absolute binary object module generation. Use of this parameter causes an absolute binary object module to be written to the named dataset after the load process is completed. Selecting AB does not imply NX (no execution). Unless NX is also selected, the loaded program begins execution after the binary is generated. Specifying AB without *adn* causes the module to be written on a dataset named $ABD, the default dataset. Some other dataset can be specified by AB=*adn*. The dataset is not rewound before or after the file is written.

If the AB parameter is omitted, no binary generation occurs.

If OVL is specified on the loader statement, the OVLDN directive replaces AB; any value specified for AB is ignored in overlay mode. Overlay loading is fully described later in this section.

MAP=*op* Map control. The MAP parameter causes the loader to produce a map of the loaded program on the specified dataset. MAP can take any of the following values:

ON Produces a block list and an entry list including all cross references to each entry

FULL Same as MAP=ON

OFF No map is produced. MAP=OFF is the default.

PART Produces a block list only. Equivalent to MAP with no value specified.

SID[='*string*']

Debug routine loading. The SID parameter indicates the system debugging routines (SID) are to be loaded with the code. These routines comprise an additional binary dataset loaded after all DN specified datasets and before any libraries.

The '*string*', if provided, is passed to SID for evaluation as a control statement. The verb and initial separator are not required. For example, SID='I=IN,ECH=ELIST.' is a proper string specification (the period is a required terminator). For a complete description of SID parameters, see the Symbolic Interactive Debugger (SID) User's Guide, CRI publication SG-0056. If only SID is specified, all keyed default SID control statement parameter values are used.

T=*tra*

Transfer name. The T parameter allows specification of an entry name where the loader transfers control at completion of the load. The T parameter also specifies the entry included in absolute binary object modules.

The entry name is a maximum of 8 characters. If no T parameter is specified, the loader begins object program execution at either the entry specified by the first encountered START pseudo from a CAL routine or at the entry of the first main program in CFT compiled routines. If no START entries are encountered, a warning message is issued and the first entry of the first relocatable or absolute module is used.

NOTE

When the SID parameter is used, the load transfer is to the system debugger, and the T parameter is ignored. If T is coded, however, a warning message is issued to the user logfile.

NX

No execution. Inclusion of this parameter inhibits execution of the loaded program.

DEB=*l*

Job Communication Block (JCB) length. The default length is 200_8. Specifying DEB without a value changes the JCB length to 3000_8.

C=*com* Compressed load. The C parameter allows control of the starting locations of modules and common blocks. An align bit is set for each relocatable module and common block that contains an ALIGN pseudo-op. Refer to the CAL Assembler Version 1 Reference Manual, CRI publication SR-0000, or to the FORTRAN (CFT) Reference Manual, CRI publication SR-0009.

C can take on any of the following values:

ON Forces the loading of each module and common block to begin at the next available location after the previous module or common block, ignoring the align bit. Equivalent to C with no value specified.

PART Forces the loading of each module and common block with the align bit set to an instruction buffer boundary.† If the align bit is not set, then that module or common block is loaded at the next available location after the previous module or common block. C=PART is the default.

OFF Forces the loading of every module to an instruction buffer boundary.† Common blocks are forced to instruction buffer boundaries only if the align bit is set.

OVL=*dir* Overlay load. The OVL parameter indicates an overlay load sequence is specified on *dir*. Overlay loading is explained in detail later in this section. If the OVL keyword is specified without a value, the loader examines the next file of $IN for an overlay load sequence. The default is no overlay load. Selecting OVL implies NX (no execution).

CNS Crack next control statement record image. This feature allows the loader to pass parameters on to the loaded program for analysis and use during execution of the loaded program. The control statement cracked follows the LDR control statement and is not available for processing by the Control Statement Processor (CSP) after processing by the loaded program.

† Instruction buffer sizes are 20_8 words for all CRAY-1 S models and 40_8 words for the CRAY X-MP.

NOTE

When the SID parameter is specified, the CNS parameter is ignored and a warning message is written to the user logfile if CNS is present. SID prompts for the control statement for the code being debugged.

NA
: No abort. If this parameter is omitted, a caution or higher level loader error causes the job to abort.

USA
: Unsatisfied external abort. When USA is specified, the loader aborts at the end if it finds one or more unsatisfied externals. A load map listing all unsatisfied externals is produced, if called for.

L=*ldn*
: Listing output. This parameter allows the user to specify the name of the dataset to receive the map output. If L=0, all output is suppressed. The default is $OUT.

SET=*val*
: Memory initialization. Variables, named and blank common blocks, and storage areas defined by DIMENSION statements are set to 0, -1, or an out-of-range floating-point value during loading. The default is SET=ZERO.

  SET=ZERO
  : Memory is set to binary zeros.

  SET=ONES
  : Memory is set to -1 (all bits set in word).

  SET=INDEF
  : Memory is set to a value that causes an out-of-range error if the word is referenced as a floating-point operand. The ones complement of each memory address is placed in the low-order 24 bits of the respective word to aid in reading register and memory dumps. An example, in octal, of the value loaded into memory word 13216 is: 0605050037740177764561.

E=*n*
: Lists error messages. This parameter indicates which level of loader-produced error messages are not to be listed. The user specifies one of five levels of severity, where *n* is the highest level to be suppressed. The default for this parameter is E=1.

Level	Type	Description
1	COMMENT	Error does not hinder program execution.
2	NOTE	Error probably hinders program execution.
3	CAUTION	Job aborts when load process completes unless NA is selected; program might not execute properly.
4	WARNING	Job aborts when load process completes unless NX is selected; program execution is not possible.
5	FATAL	Job aborts immediately.

Example:

E=2 suppresses COMMENT and NOTE messages and allows CAUTION, WARNING, and FATAL messages to appear. FATAL messages are never suppressed.

I=*sdir* Selective load. Modules from other datasets can be loaded according to a set of directives. *sdir* indicates the dataset containing the directives. If the I keyword is specified without a value, the directives are taken from the next file of $IN. The selective load directives are described later in this section.

NOECHO Suppress writing the current control statement to the user logfile (that is, the conrol statment which invoked the actual loading into memory will not be written to the logfile).

SECURE Define each dataset created during this job step to be *secure* (that is, to be released during job advancement unless specifically overridden with a F$DSD operating system request).

GRANT Grant the privileges defined as parameters if this module is loaded from the System Directory (SDR). (These privileges will be merged with the users' only for the duration of the job step.) The following parameters are defined if security is enabled:

SCISPT	Allow ISP testing
SCRDSC	Read DSC page
SCSPOL	SAVE/ACCESS/DELETE/LOAD/DUMP spooled dataset
SCLUSR	Load user dataset
SCDTIM	Dump time request
SCQSDT	Dequeue/queue SDT requests
SCUPDD	Access user dataset for PDSDUMP
SCACES	Access user-saved dataset without passwords
SCQDXT	LINK/MODIFY DXT requests
SCENTR	ENTER option on ACCESS
SCNVOK	Invoke job class structure
SCDUMP	Allow F$DJA requests anytime
SCPRIV	Allow special system requests
SCSYSPRG	Allow system programmer functions such as F$PROF and F$CMEM
SCURID	Allow use of reserved ID in interjob communication
SCERCH	Allow F$DRIVER requests
SCERQM	Allow SDT queue manipulation
SCMLOG	Allow a user to send messages to another user's logfile
SCSYSJ	Allow a job to be a system job

BC=*bc* Blank common. *bc* specifies the decimal number of words to be added to the size of blank common when the program is loaded for execution. The default is 0.

PAD=*pad* Pad. *pad* specifies the decimal number of words of unused space to be made available in the job when the program is loaded for execution. After the program is loaded with its requested extra space the job is placed in user-managed field length reduction mode for the duration of the job step. The default is 0.

NORED No field length reduction. Before the program is loaded the job is placed in user-managed field length reduction mode for the duration of the job step.

STK[=*initial size*[:*increment*]]

Initializes for stack processing. STK is a run-time memory management parameter.

initial size indicates the initial size of a stack in number of words. An installation parameter defines the default value. If the *initial size* value is less than 128, LDR substitutes the default value.

increment specifies the size of additional segments to a stack (in number of words) if a stack overflows. An installation parameter defines the default value. A value of 0 indicates that overflow is prohibited.

MM[=*initial size*[:*increment*]]

Initializes for managed memory processing. The values assigned to MM specify the number of words available to the heap manager.

initial size indicates the number of words initially available to the heap manager. An installation parameter defines the default value. The loader changes the specified value if the heap is not allowed to grow, and if there is no room for heap and stack overhead.

increment specifies the minimum size, in words, of a request to the operating system for additional memory if the heap overflows. Zero means that the size of the heap is fixed. An increment other than zero cannot be specified if the heap is before blank common. An installation parameter defines the default value. If the BEFORE value is specified for MMLOC, then the default value is 0.

MMEPS=*epsilon*

epsilon is the smallest block that can be left on the list of available space in the heap. If a request for additional memory from the heap is made by the run-time routines, and the request leaves a memory fragment of less than *epsilon* words, the additional words are given to the request. The value must be at least 2. An installation parameter defines the default value.

$MMLOC=\begin{Bmatrix} AFTER \\ BEFORE \end{Bmatrix}$

Specifies the location of the heap. AFTER specifies that the heap is located after blank common. It is also the default. If the heap is located before blank common, BEFORE is specified.

LOAD MAP

Each time the loader is called, the user has the option of requesting a listing that describes where each module is loaded and what entry points and external symbols are used for loading. This listing is called a load map.

The user specifies the contents of the map or the dataset to receive the map by setting parameters of the LDR control statement to the desired values. The MAP parameter of the LDR control statement allows the user to specify the contents of the map requested. MAP=ON or MAP=FULL produces a block list and an entry list. The block list gives the names, beginning addresses and lengths of the program and subroutines loaded on this loader call; the entry list includes all cross references to each entry. MAP=PART supplies a partial map, that is, the block map only.

The load map is printed when requested even if fatal errors abort the load. In this case, the map contains only those modules loaded up to the point where the fatal load error occurred.

Figure 14-1 illustrates the load map generated by the following LDR statement:

```
LDR,DN=$BLD:LOAD2,LIB=MYLIB,MAP=FULL,MM=16000:4000,STK=1280:128
```

The block list consists of items 1 through 16 in figure 14-1; the entry list includes items 17 through 21.

```
                                  1                   2                             3           4
TOTAL: 1321                    LDRMAP            LDR X.14  84251    09/24/84               11:54:11    PAGE 1
                            5
                  RELOCATABLE LOAD

LOAD TRANSFER IS TO  6   AT  ( 7 )

DATASET   BLOCK    ADDRESS    LENGTH    DATE        OS REV     PROCSSR    VER.        Comment
  8          9        10         11       12          13         14        15            16
          *SYSTEM      0       200
$BLD      LDRMAP     200       1321     09/24/84    COS X.14   CFT 1.13   09/21/84
LOAD2     ABCDEFGH  1521       36       09/24/84    COS X.14   CFT 1.13   09/21/84
MYLIB     X1        1557       41       09/24/84    COS X.14   CFT 1.13   09/21/84
          X2        1620       41       09/24/84    COS X.14   CFT 1.13   09/21/84
          .
   17     .         18          19         20                 21
MODULE NAME      ENTRIES     ENTRY VALUE REF. MODULE         ABSOLUTE     REFERENCES

     LDRMAP       LDRMAP        717a
     ABCDEFGH     ABCDEFGH     1525a     LDRMAP    1425a
     X1           X1           1570a     ABCDEFGH  1531a
     .
     .
     .
                  NLERP%       3234a
     $FDP         $FDP         4640      $WUT      10603b
     $WFD         $WFI         5451a     LDRMAP    1410a        1416d
      22
*** MANAGED MEMORY STATISTICS ***
INITIAL STACK SIZE:              1280(10)  2400(8) WORDS
STACK INCREMENT SIZE:            128(10), 200(8) WORDS
INITIAL MANAGED MEMORY SIZE:     16000(10),        37200(8)  WORDS
MANAGED MEMORY INCREMENT SIZE:   4000(10),          7640(8)  WORDS
MANAGED MEMORY EPSILON:                            2(10),
2(8) WORDS
BASE ADDRESS OF MANAGED MEMORY/STACK:   15566(10),           36316(8)
WORDS
MANAGED MEMORY/STACK LOCATION: AFTER BLANK COMMON

*** LOAD IMAGE STATISTICS ***      23
ABSOLUTE BINARY LENGTH: 31438(10),     75316(8)  WORDS
PROGRAM IMAGE: FWA =             200(8), LWA =     75516(8)
```

Figure 14-1. Example of a load map

1. Job name from the JOB control statement

2. Loader level and Julian date of assembly of the loader

3. Date and time of loader execution

4. Page number

5. Load type; either relocatable, absolute, or overlay

6. Entry name to which initial transfer is given

7. Entry address where initial transfer is made

8. Name of load or library dataset containing modules to be loaded

9. Names of blocks loaded from the named dataset. These are common blocks (identified by the slashes around their names, for example, /LABEL/) are names of program blocks.

   *SYSTEM is always the first block listed in a relocatable load. It consists of the first 200 (octal) words of the user field, which is reserved for the Job Communication Block (JCB). For an absolute load, *SYSTEM is not allocated. Therefore, the CAL user must set the origin to 200 (octal) via an ORG pseudo instruction to allow space for the JCB. If this is not done, the job aborts.

   Blank common, indicated as //, is allocated last and appears at the end of the list (if it has been defined).

10. Octal starting address of the block

11. Octal word length of the block

12. Date the object module was generated

13. Operating system revision date at the time the object module was generated

14. Name and revision level of the processor that generated the object module

15. Revision date of the processor that generated the object module

16. Comment (if any) from CAL COMMENT pseudo included in the load module

17. Name of program block referenced

18. Entry points in the program block

19. Word address, parcel address, or value of each entry point

20. Module name of reference to each entry point

21. Absolute parcel addresses of references to each entry point. Eight references are listed per line; some entry points have no references.

22. Managed memory statistics. The numbers in parentheses indicate the base: decimal (10) and octal (8).

23. Actual length of the binary; the minimum amount of memory required to load the program. FWA is the first word address of the load image. LWA is the last word address of the load image. The numbers in parentheses indicate the base: decimal (10) and octal (8).

SELECTIVE LOAD

If the I keyword is present on the LDR control statement, one or more INCLUDE and/or EXCLUDE directives are examined in the specified dataset.

Formats:

INCLUDE,SDN=*sdn*,FN=*fn*,MOD=md_1:md_2:...:md_{50}.

EXCLUDE,SDN=*sdn*,FN=*fn*,MOD=md_1:md_2:...:md_{50}.

Parameters are in keyword form.

SDN=*sdn* Name of dataset containing modules to be selectively loaded. If SDN is specified without a value, the first dataset specified on the DN parameter of the LDR statement is the default. If the SDN parameter is omitted, an error message results, and the directive is skipped; the load does not abort. The SDN and FN parameters must refer to the same dataset.

FN=*fn* File number of the specified dataset. A number from 0 through 7. *fn* refers to the file by its numerical position in SDN or in the DN parameter of the LDR statement.

For example, if DN=D1:D1:D2, the first file of D1 has an *fn* of 0, and the second file of D1 has an *fn* value of 1. If FN is specified without a value, the default is 0. If FN is omitted, the whole of *sdn* is searched for the correct module; a message is issued for a complete *sdn* search. The SDN and FN parameters must refer to the same dataset.

To load a module from the first file of D1, the directive can include the parameter FN=0; however, if FN is specified without a value, the default is to load a module from the first file.

MOD=*md* Module name or entry point to a module to be included or excluded from the load. Up to 50 modules can be specified; the modules must be separated by colons. If the MOD parameter is omitted, an error message results, and the directive is skipped.

Example: Given the LDR statement

```
LDR,DN=D1:D1:D2,...,I.
```

A directive to load a module from the second file of dataset D1 includes the following directive in the next file of $IN:

```
INCLUDE,SDN=D1,FN=1,MOD=... .
```

Selective load messages are never suppressed.

PARTIALLY RELOCATED MODULES

When a binary module is defined as a relocatable overlay, the loader can generate an image of the module that has been only partially relocated. The image of the binary module contains sufficient information for a user program to relocate all address references within the module program block according to the actual address where the user program determines the module should be executed.

The relocatable overlay is useful because program modules are generated so that a common memory pool can execute the overlay and any of several overlays can execute at any address within the pool.

GENERATION OF RELOCATABLE OVERLAYS

The CAL assembler defines a module as a relocatable overlay at assembly time with the MODULE pseudo-op.

Format:

Location	Result	Operand
ignored	MODULE	*type*

Parameters:

type A keyword parameter identifying the type of module being defined. RELOCOVL is the only type currently available.

When the relocatable overlay is defined by the assembler, COS sets a special flag in the Program Description Table (PDT) for use by the relocatable loader.

The loader, recognizing that the current module being loaded is a relocatable overlay, performs limited relocation of the address references in the module. That is, all references to labeled common blocks and all references to entry points defined within other modules are adjusted according to the address where the other module resides in the memory image being constructed. References to blank common are illegal. It is also illegal for any other module to make any reference to any entry point defined to be within the relocatable overlay module. References from within the module to addresses within the module are not adjusted at this time. Instead, a copy of the necessary Block Relocation Table (BRT) entries is included in the memory image of the module. All BRT entries not needed for satisfying internal references are deleted.

The absolute memory image of the program constructed by the loader contains the loaded programs, including all relocatable overlay modules.

The relocatable overlays are physically located at the end of the memory image; all nonrelocatable overlay modules are loaded contiguously in the order they are encountered. Relocatable overlay modules can appear at any point in the load sequence and can be contained in libraries. The loader moves modules in memory as required to order the relocatable overlays at the end of the image. This placement of the overlays makes it possible for a user program to locate the images of each overlay and to copy the overlays to mass storage, if it is desired, in order to make the memory space used by the overlay images available for use by the program.

MEMORY LAYOUT WHEN RELOCATABLE OVERLAYS EXIST

When the loader has detected the existence of one or more relocatable overlays, memory is laid out in the following manner:

1. All nonrelocatable modules, in the order they are encountered on load datasets or in libraries

2. Labeled common blocks interspersed among the nonrelocatable modules so that a labeled common block precedes the absolute image of the first block encountered which defines the block

3. All labeled common blocks defined first within a relocatable overlay module and not defined within any other type of module

4. Images of all relocatable overlays in the order they are encountered on load datasets or in libraries

5. Unsatified external (USX) program which is the loader's internal program for processing unsatisfied external references

6. Blank common if defined by any program module

Note that the placement of USX and blank common can defeat the purpose of relocatable overlays, since the overlay images must remain reserved. With proper care, the program can use the space occupied by the overlay images for internal tables and other data with nonallocated space.

MEMORY LAYOUT OF A RELOCATABLE OVERLAY IMAGE

When the loader completes constructing the image of the complete program being loaded, the relocatable overlay portions have a different structure than do the nonrelocatable overlay portions. Normal modules are loaded as an absolute image with all loader-related tables removed. All address references, both internal to the module and to other modules, are adjusted so that the code executes correctly. If the C parameter is specified when the loader is called into execution, individual modules can begin immediately after the previous module, or they can begin at the next 16-word (decimal) boundary.

Because relocatable overlay modules are expected by the loader to be moved to a different address for execution, the C specification has no meaning to a relocatable overlay module, and the first and subsequent such modules begin immediately after the last word of the previous module.

Relocatable overlay module images also contain loader-relocated tables. These tables are required so that the user program can adjust address references within a relocatable overlay when it has determined the address where the overlay will execute. The tables are:

PDT Program Description Table
TXT Text Table
BRT Block Relocation Table

The PDT contains information regarding the number of entry points defined and the number of blocks and external references. The TXT contains a count of the words in the actual image of the code, followed by the semi-absolute image of the code. The BRT contains information necessary for adjusting address references within the module. If the user program wants to write the overlays to mass storage, the information in the PDT can be used to construct a directory or similar table for locating specific overlays or entry points, and then can be discarded. TXT and BRT must be retained in the mass storage copy for future relocation of address references.

OVERLAYS

Very large programs might not fit in the available user memory space or might not use large portions of memory while other parts of the program are in execution. For such programs, the COS relocatable loader includes the ability to define and generate *overlays*, separating modules that the user creates and then calling and executing as necessary.

Two types of overlays are available.

- *Type 1 overlays* are generated by using the directives ROOT, POVL, and SOVL. Two levels of overlays in addition to the root overlay are allowed with calls to a maximum of 999 adjacent overlays.

- *Type 2 overlays* are generated by using the directive OVLL. Ten levels of overlays in addition to the root overlay are allowed with calls to a maximum of 63 adjacent overlays.

The overlay loader can also generate a partially relocated module, referred to as a relocatable overlay. Relocatable overlays have been fully described earlier in this section.

The overlay structure, rules for overlay generation, and overlay calls for both types are described in this section. The control statements used to generate the overlay and the directives common to both types of overlays are described first. Specific rules for generation of Type 1 and Type 2 overlays are described separately in the following subsections.

Overlay generation consists of a load operation in which the loader performs relocatable loading and writes the resulting binary image to disk. One named absolute binary record is written per root and each overlay.

If the LDR control statement has the parameter OVL=*dir*, the loader finds the overlay generation directives on the named dataset, *dir*. If no dataset is given (that is, OVL), then the loader reads overlay generation directives from $IN.

Format:

```
LDR,...,OVL=dir,... .
```

OVERLAY DIRECTIVES

An overlay directive consists of a keyword and a parameter. A blank, comma, or open parenthesis must separate the keyword from the parameter. A period, closed parenthesis, or two consecutive blanks serve as the terminator. A caret at the end of the directive line indicates that the next line is a continuation of the current directive. The caret cannot be preceded by a blank; it must immediately follow the last character of the line.

FILE directive

The FILE directive indicates the dataset, *dn*, containing the routines to be loaded. This directive's function is similar to that of the DN parameter on the LDR control statement. It is generally the first directive on the directives dataset but appears at any time and as often as necessary thereafter. If no FILE directive appears, the loading proceeds from the dataset specified on the DN parameter of the LDR control statement. If that too has been omitted, loading initially occurs from $BLD. This directive is common to both overlay types.

Format:

```
FILE,dn.
```

OVLDN directive

The function of this directive is similar to that of the AB parameter on the LDR control statement. This directive names the dataset, *dn*, on which overlays are written. The *dn* parameter must be present. If no OVLDN directive is present, the default overlay binary dataset ($OBD) is assigned. All overlays generated following an OVLDN directive reside as separate binary records on dataset *dn*. OVLDN directives appear as often as desired. This directive is common to both overlay types.

Format:

```
OVLDN,dn.
```

SBCA directive

The SBCA directive sets the blank common starting address to the specified address. This directive allows the user to place blank common after all load modules in the current overlay structure. The address specified must be larger than any address used in the overlay structure. This directive must appear before any overlay generation directive, such as ROOT or OVLL.

Format:

```
SBCA,address.
```

where *address* is the octal address assigned to blank common.

TYPE 1 OVERLAY STRUCTURE

Each Type 1 overlay is identified by a pair of decimal numbers, each from 0 through 999. There must be one and only one root overlay; its level numbers are (0,0). This root remains in memory throughout program execution. Primary overlays all have level numbers (*n*,0) where *n* is in the range 1 through 999.

Primary overlays are called at various times by the root and are loaded at the same address immediately following the root. A secondary overlay is associated with a specific primary overlay, and can be called only by the corresponding primary overlay. The secondary level numbers are (*n*,*m*), where *n* is the primary level, and *m* is in the range 1 through 999. All secondary overlays associated with a given primary (that is, the same *n*) are loaded at the same address immediately following that primary.

Only the root, one primary overlay, and one secondary overlay can be in memory at one time.

Figure 14-2 is a diagram of a sample Type 1 overlay loading. The primary and secondary overlays are shown in time sequence. The sequence of generation does not imply that the routines are loaded into memory in the same sequence or that they remain in memory for a set period of time when they are executed.

All external references must be directed toward an overlay nearer to the root. For example, overlay (1,0) can contain references to the root (0,0) but not to overlay (1,1). Overlay (1,1) can contain references to both (1,0) and (0,0).

The loader places named common before the routine that first references it. All named common references must be directed toward a lower level routine. The lowest level routine with a named common block must contain data statements for that block.

For example, in figure 14-2,

MAIN	Can reference named common A only
SUB1 and SUB2	Can reference named common A and B only
TEST	Can reference named common A, B, and C

The loader allocates blank common immediately after the first overlay where it is declared. If blank common is declared in the root overlay (0,0), it is allocated at the highest address of the root overlay and is accessible to all overlays. If blank common is first declared in primary overlay (1,0) and not declared in the root (0,0), then it is accessible only to the (1,*x*) overlays. Allocation and placement of blank common is also manipulated by the user through the SBCA directive.

JCHLM is set to the highest address of the root overlay before loading. If a subsequent overlay module requires additional memory, JCHLM is reset to the highest address of that module.

TYPE 1 OVERLAY GENERATION DIRECTIVES

The overlay generation directives define the structure of the overlay. Included in this class are the ROOT, POVL, and SOVL directives.

ROOT directive

This directive defines programs, subroutines, and/or entry points comprising the load from *dn*. For programs written in CAL, list each entry referenced. FORTRAN programs need the program name only. All members for this directive reside on the same dataset, *dn*, as defined by the FILE directive.

Format:

ROOT,*member*$_1$,*member*$_2$,*member*$_n$.

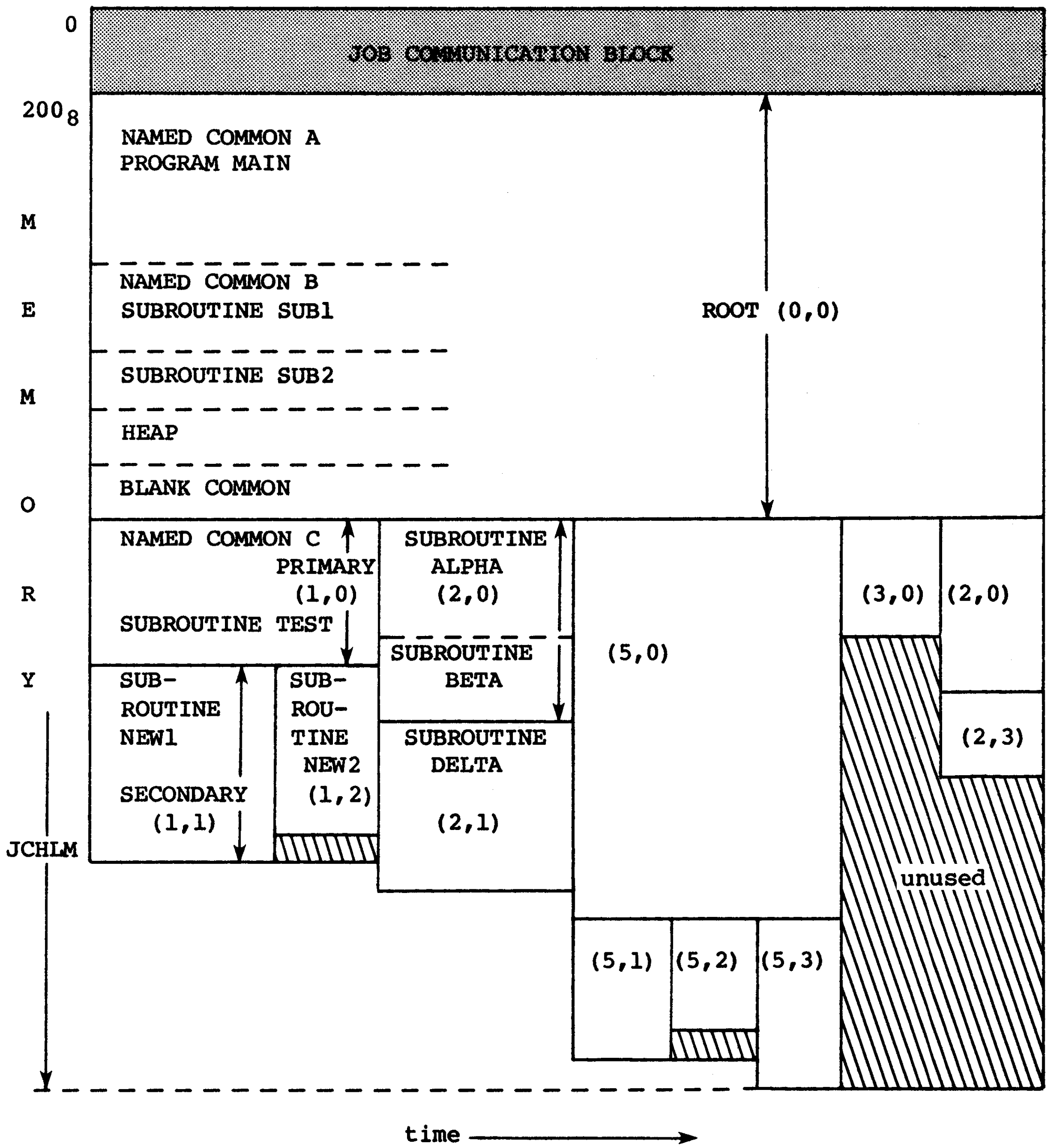

Figure 14-2. Example of Type 1 overlay loading

POVL directive

This directive causes relocatable loading of the named blocks to the primary overlay with the name *plevel*:000. The size of the root determines the base location. All members for this directive reside on the same dataset, *dn*. The first member in the list is the one that receives control when the overlay is loaded. For routines written in CAL, the first entry point of the first routine receives control.

Format:

POVL,*plevel*,*member*$_1$,*member*$_2$,...,*member*$_n$.

where *plevel* is between 1 and 999.

SOVL directive

This directive causes relocatable loading of the named blocks to the secondary overlay with the name *plevel*:*slevel*. The length of POVL (*plevel*:000) determines the base location. All members for this directive reside on the same dataset, *dn*. The first member in the list is the one that receives control when the overlay is loaded. For routines written in CAL, the first entry point of the first routine receives control.

Format:

SOVL,*slevel*,*member*$_1$,*member*$_2$,...,*member*$_n$.

where *slevel* is between 1 and 999.

Generation directive example

In the following example,

DSET1 contains routines THETA, TEST, GAMMA, SUB1, MAIN, SUB2.

DSET2 contains routines NEW2, ALPHA, OVER, NEW1, DELTA, EPSILON, SIGMA, BETA.

Format of the control statement that initializes overlay generation:

```
LDR,...,OVL=OVLIN,....
```

Dataset OVLIN contains the following directives:

FILE,DSET1.	Loader selectively loads from dataset DSET1.
OVLDN,LEV00.	The following overlay modules are written to the dataset LEV00.
ROOT,MAIN,SUB1 ,SUB2.	The absolute binary of MAIN,SUB1,SUB2 is written as the first record on dataset LEV00.
POVL,1,TEST.	The binary of TEST is named 001:000 and is binary record 2 on dataset LEV00.
FILE,DSET2.	Loader selectively loads from dataset DSET2.
SOVL,1,NEW1.	The binary of NEW1 is named 001:001 and is binary record 3 on dataset LEV00.
OVLDN,LEV12.	The subsequent overlay modules are written to the dataset LEV12.
SOVL,2,NEW2.	The binary of NEW2 is named 001:002 and is binary record 1 on dataset LEV12.
POVL,2,ALPHA,BETA.	The binary of ALPHA,BETA is named 002:000 and is record 2 on dataset LEV12.
.	
.	
.	
eof	End of overlay load sequence

TYPE 1 OVERLAY GENERATION RULES

1. Overlay members are loaded from datasets named in FILE directives. Members are searched for in the most recently mentioned dataset only. In the absence of a FILE directive, members are loaded from the dataset specified on the LDR control statement. If that is also omitted, loading will initially occur from $BLD. Currently, the relocatable modules of all members for any overlay level must reside on the same file.

2. The overlays are generated in the order of the directives.

3. There must be one and only one root.

4. Level hierarchy must be maintained. The root overlay must be generated first; hence the ROOT directives appear first. Following the root generation, a primary overlay (POVL) is generated. No limitation is placed on which primary overlay number (*plevel*) is generated; however, all secondary overlays (SOVL) associated with the *plevel* must follow. The secondary overlay *slevels* can be generated in any order following their respective primary level.

5. An end-of-file in the directives file ends the input of overlay directives; hence overlay generation.

6. Any directive other than FILE, OVLDN, SBCA, ROOT, POVL, or SOVL causes a fatal error.

7. The list of members can be continued to another line by using a caret immediately following the last nonblank character at the end of the directive line. The ∧ does not replace a separator and must not appear within a member name.

8. Any number of lines can be used to name the members of an overlay.

9. A secondary overlay can only be called by the corresponding primary overlay.

TYPE 1 OVERLAY EXECUTION

A control statement call of the dataset containing the ROOT overlay initiates its loading and execution. If no OVLDN directives are used before generating the ROOT, the dataset $OBD contains the ROOT overlay.

The following sequence executes the root overlay after generation:

LDR,...,OVL=*dir*,... .
$OBD.

During overlay generation the members are loaded from the FILE dataset in the order they appear on the dataset, regardless of their order of appearance in the members list. The entry for POVL and SOVL overlays is defined by the first member listed on the generation directive. Control is transferred to this address after loading by the $OVERLAY routine during program execution. The ROOT entry is named using the T parameter on the LDR control statement.

The user calls for the loading of overlays from within the program, and the method by which they are called depends on the program language in use (FORTRAN or CAL). OVERLAY is a subroutine of the root overlay and is loaded into memory with the root.

FORTRAN language call

A FORTRAN program calls for the loading of overlays as follows:

```
CALL OVERLAY(dn,level1,level2,r)
```

dn — Dataset name or unit number that contains the file to be skipped. Must be a character constant, integer variable, or an array element containing Hollerith data of not more than seven characters.

$level_1$ — Primary level number of the overlay

$level_2$ — Secondary level number of the overlay

r — An optional recall parameter. If the user wishes to re-execute an overlay without reloading it, 6LRECALL is entered. If not currently loaded, it will be loaded.

CAL language call

A sample call sequence from a CAL program is as follows:

Location	Result	Operand
	EXT	OVERLAY
	.	.
	.	.
	.	.
	CALL	OVERLAY,(OVLDN,PLEV,SLEV)
	.	.
	.	.
	.	.
OVLDN	CON	A'LEV12'L
PLEV	CON	2
SLEV	CON	0

where OVLDN is the address of the dataset name, PLEV is the address of the primary level, and SLEV is the address of the secondary level. If recall is desired, the address of the literal 'RECALL' is transmitted as the fourth argument.

Example:

Location	Result	Operand	Comment
1	10	20	35
	CALL	OVERLAY,(OVLDN,PLEV,SLEV,RECL)	
	.	.	
	.	.	
	.	.	
RECL	CON	'RECALL'L	

For both FORTRAN and CAL language calls, during execution of the ROOT(0,0) program MAIN, the statement

CALL OVERLAY(5LLEV12,2,0) or the above CAL sample call

causes OVERLAY to search dataset LEV12 for the absolute binary named 002:000. OVERLAY positions the dataset LEV12 to the location of the absolute binary named 002:000 using information supplied by the loader, loads the overlay, and transfers control to the first member specified on the POVL or SOVL directive. After execution of the overlay, control returns to the statement in MAIN immediately following the CALL statement. Following the load, dataset LEV12 is positioned immediately after the end of record for the overlay (2,0). If overlay (2,0) is not on dataset LEV12, a fatal error results.

Placing a call for a secondary overlay for which the corresponding primary overlay is not already loaded causes a fatal error. A fatal error also results if the primary and secondary overlays are not both on the named *ovldn*.

TYPE 2 OVERLAY STRUCTURE

Figure 14-3 shows an example of a tree structure of the Type 2 overlay. There is only one root overlay, and its level number is 0. The root overlay remains in memory during program execution and calls only level 1 overlays. Only one branch is in memory at any time. Overlay (2,1) under overlay (1,1) is different from the (2,1) under (1,5). Moreover, overlay (2,1) under overlay (1,1) can be called only by overlay (1,1)

Figure 14-4 shows a sample Type 2 overlay loading diagram. The overlays are shown in time sequence. The sequence of generation does not imply that the programs are loaded into memory in the same sequence or that they remain in memory for a set period of time when they are executed.

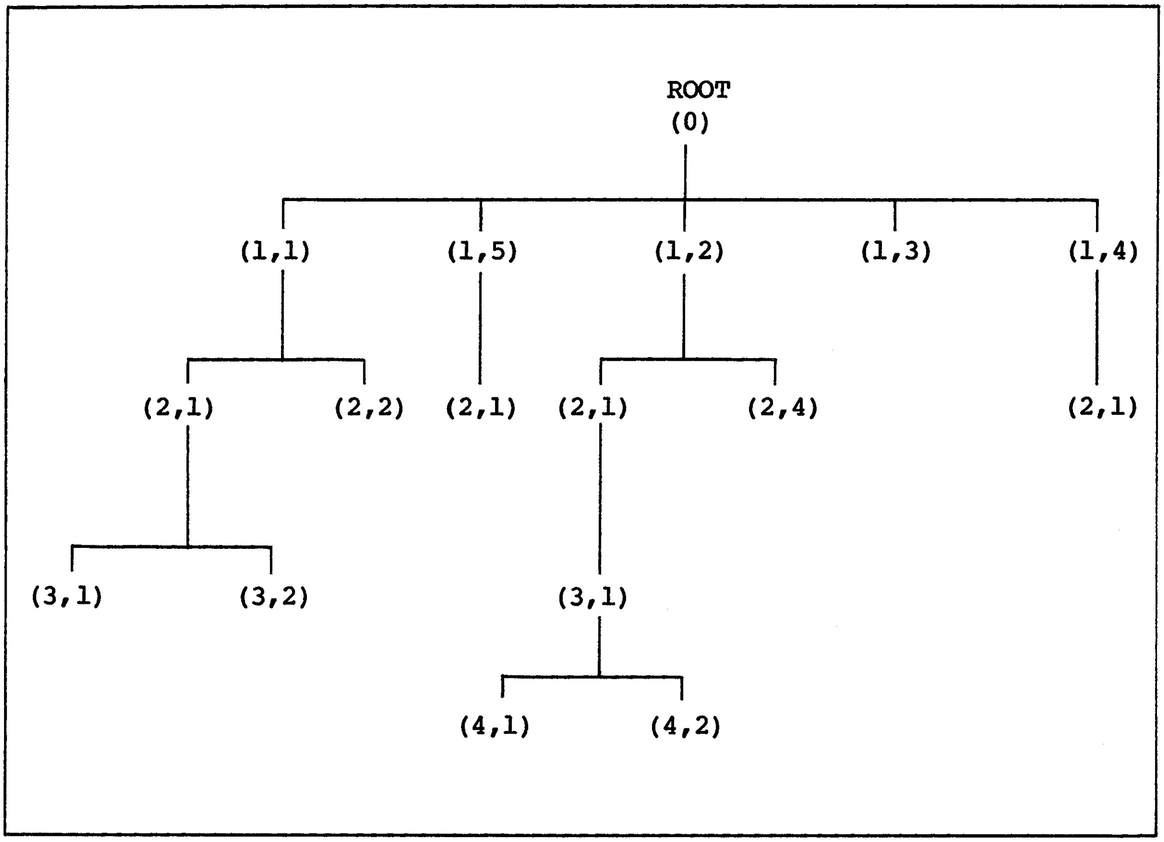

Figure 14-3. Example of the Type 2 overlay tree

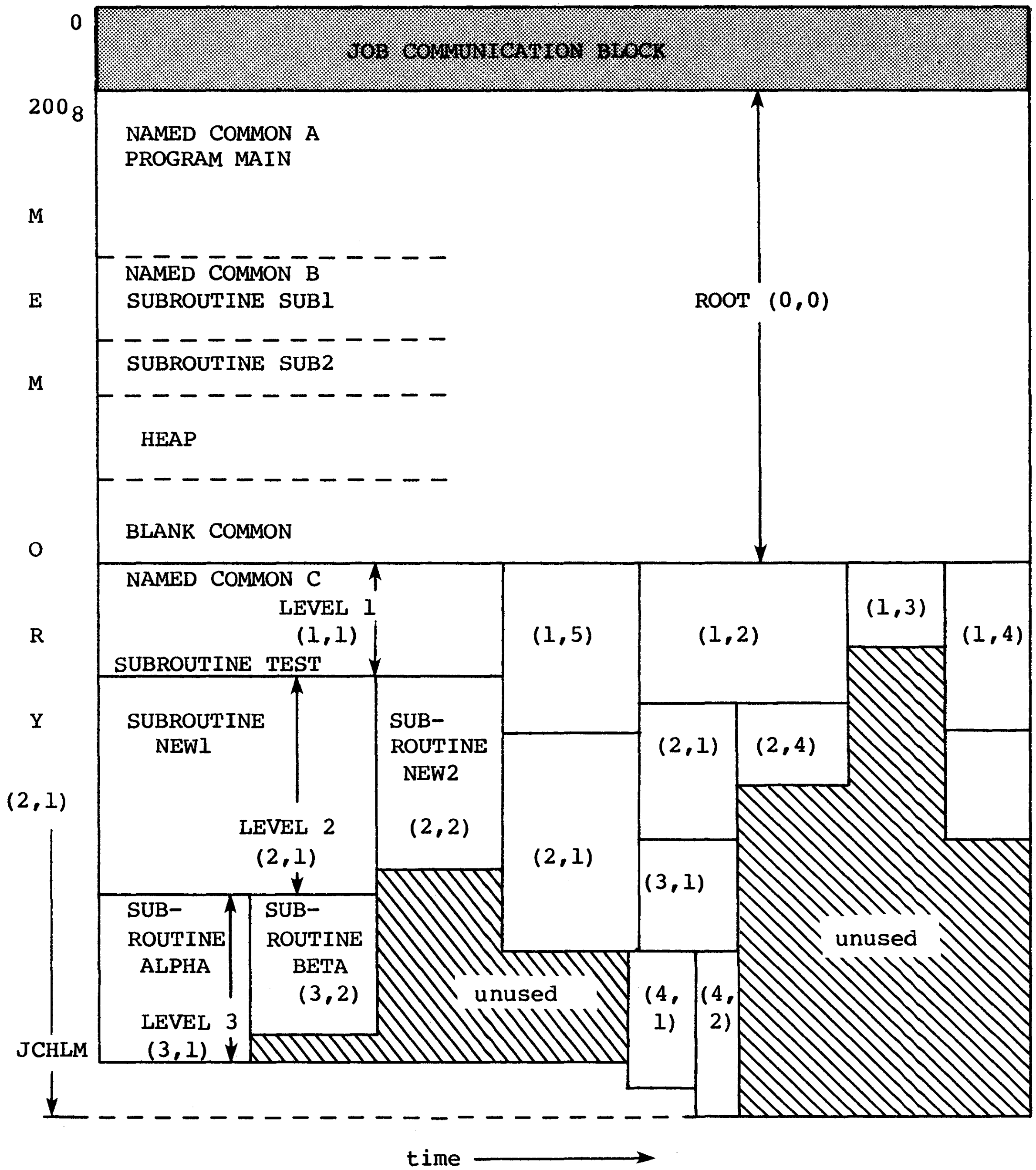

Figure 14-4. Example of Type 2 overlay loading

Level 1 overlays are called at various times by the root overlay. Each call loads the named overlay at the same address, immediately following the location of the root. The first level overlay must be called by the root. Each upper level overlay must be called by the associated overlay at the adjacent lower level. A hierarchy exists among overlay levels; an upper level overlay is subordinate to the proximate lower level overlay. An upper level overlay associated with overlay (2,1) might be (3,2), (3,3) or (3,4).

An overlay can call into memory any overlay in the next higher level; it cannot call an overlay more than one level above it in the hierarchy. For example, overlay (2,1) can call (3,1) through (3,63), but it cannot call (4,1). Each call for an overlay loads the named overlay at the same address location immediately following the location of the calling overlay. Only the root and one overlay at each level can be in memory concurrently.

All external references must be directed toward an overlay nearer the root overlay. Overlay (1,1) can contain references to the root overlay but not to overlay (1,2) or overlay (2,1). The (2,1) overlay can reference externals in both the (1,1) overlay and the root overlay.

The loader places named common blocks before the routine that first references it. All named common references must be directed toward a lower level routine (toward the root overlay). If blank common is declared in the root overlay, it is allocated at the highest address of the root and is accessible to all overlays. If blank common is declared first in a level 1 overlay, for example, and is not declared in the root overlay, it is accessible only to level 1 and upper level overlays.

JCHLM is set to the highest address of the root overlay before loading. If a subsequent overlay module requires additional memory, JCHLM is reset to the highest address of that module.

TYPE 2 OVERLAY GENERATION DIRECTIVE

The Type 2 overlay directive defines the structure of the overlay within the directive format.

OVLL directive

This directive causes relocatable loading of the named blocks of an overlay. The size of the lower level overlays in the group determines the base location. All members for this directive reside on the same dataset, *dn*, specified by the FILE directive. The first member in the list is the one that receives control when the overlay is loaded. For programs written in CAL, the first entry point of the first routine receives control.

Format:

```
OVLL,level,number,member1,member2,...,membern.
```

level Either a level number of the overlay (1 through 10), or the root phase (0). If the root phase is being generated, *number* must be omitted.

number Number of the overlay (1 through 63) within the level

member Module names for the individual overlays

Generation directive example

In the following example,

DSET1 contains routines THETA, TEST, GAMMA, SUB1, MAIN, SUB2.

DSET2 contains routines NEW2, ALPHA, OVER, NEW1, DELTA, EPSILON, SIGMA, BETA.

Format of the control statement that initializes overlay generation:

```
LDR,...,OVL=OVLIN,...
```

Dataset OVLIN contains the following directives:

FILE,DSET1.	Loader selectively loads from dataset DSET1.
OVLDN,LEV00.	The following overlay modules are written to the dataset LEV00.
OVLL,0,MAIN,SUB1, SUB2.	The absolute binary of MAIN,SUB1,SUB2 is the first record on dataset LEV00.
OVLL,1,1,TEST.	The binary of TEST is binary record 2 on dataset LEV00.
FILE,DSET2.	Loader selectively loads from dataset DSET2.
OVLL,2,1,NEW1.	The binary of NEW1 is binary record 3 on dataset LEV00.
OVLDN,LEV12.	The subsequent overlay modules are written to the dataset LEV12.

OVLL,2,2,NEW2.	The binary of NEW2 is binary record 1 on dataset LEV12.
OVLL,3,1,ALPHA.	The binary of ALPHA is binary record 2 on dataset LEV12.
OVLL,3,2,BETA.	The binary of BETA is binary record 3 on dataset LEV12.
.	
.	
.	
eof	End of overlay load sequence.

TYPE 2 OVERLAY GENERATION RULES

1. Overlay members are loaded from datasets named in FILE directives. Members are searched for in the most recently mentioned dataset only. In the absence of a FILE directive, members are loaded from the dataset specified on the LDR control statement. If that is also omitted, loading initially occurs from $BLD.

2. The overlays are generated in the order of the directives.

3. There must be one and only one root per dataset.

4. Level hierarchy must be maintained. The root overlay must be generated first. Following the root generation, a first level overlay is generated. No limitation is placed on which overlay number is generated; however, all overlays associated with that first level overlay must follow. The overlays can be generated in any order; the same restrictions apply for all levels of overlays (1 through 10).

5. The first level overlay must be called by the root. An overlay can call into memory any overlay in the next higher level. However, an overlay cannot call an overlay that is more than one level above it in the hierarchy.

6. An end-of-file ends the input of overlay directives.

7. Any directive other than FILE, OVLDN, SBCA or OVLL causes a fatal error.

8. The list of members can be continued to another line by using a caret immediately following the last character at the end of the directive line (that is, no blanks). The caret does not replace a separator and must not appear within a member name.

9. Any number of lines can name the members of an overlay.

TYPE 2 OVERLAY EXECUTION

A control statement call of the dataset containing the root overlay initiates the root overlay's loading and execution. If no OVLDN directives are used before generating the root, the dataset $OBD contains the root overlay. All overlays reside on the datasets specified on the overlay directives. The entry for higher level overlays is defined by the first member listed on the generation directive. Control is transferred to this address after loading by the $OVERLAY routine during program execution. The root entry is named using the T parameter on the LDR control statement.

The following sequence executes the root overlay after generation:

```
LDR,...,OVL=dir,... .
$OBD.
```

When the program is to be executed, the root overlay is brought into memory as a result of a control statement call in the job deck. Thereafter, additional overlays are called into memory by the executing program. Overlay loading allows any overlay to call for the loading of an adjacent upper level overlay.

The user calls for the loading of Type 2 overlays from within the program, and the method by which they are called depends on the program language in use (FORTRAN or CAL). OVERLAY is a subroutine of the root overlay and is loaded into memory with the root.

FORTRAN language call

A FORTRAN program calls for the loading of Type 2 overlays as follows:

CALL OVERLAY(*n*L*dn*,*level*,*number*,*r*)

n	Number of characters in the name
L	Left-adjusted; zero-filled.
dn	Dataset name where this overlay resides
level	Level number of the overlay
number	Number of the overlay within the level
r	Optional recall parameter. If the user wishes to re-execute an overlay without reloading it, 6LRECALL is entered. If not currently loaded, it will be loaded.

CAL language call

A sample call sequence from a CAL program is as follows:

Location	Result	Operand
	EXT	OVERLAY
	.	.
	.	.
	.	.
	CALL	OVERLAY,(OVLDN,PLEV,SLEV)
	.	.
	.	.
	.	.
OVLDN	CON	A'LEV12'L
PLEV	CON	2
SLEV	CON	0

where OVLDN is the address of the dataset name, PLEV is the address of the primary level, and SLEV is the address of the secondary level. If recall is desired, the address of the literal 'RECALL' is transmitted as the fourth argument.

Example:

Location	Result	Operand	Comment
1	10	20	35
	CALL	OVERLAY,(OVLDN,PLEV,SLEV,RECL)	
	.	.	
	.	.	
	.	.	
RECL	CON	'RECALL'L	

For both FORTRAN and CAL language calls, during execution of the ROOT program MAIN, the statement

CALL OVERLAY(5LLEV12,1,2), or above CAL sample call

causes OVERLAY to search dataset LEV12 for the absolute binary named 2. OVERLAY positions the dataset LEV12 to the location of the absolute binary named 2 using information supplied by the loader, loads the overlay, and transfers control to the first member specified on the OVLL directive. After execution of the overlay, control returns to the statement in MAIN immediately following the CALL statement. Following the load, dataset LEV12 is positioned immediately after the end of record for the overlay 2. If overlay 2 is not on dataset LEV12, a fatal error results.

OVERLAY GENERATION LOG

When MAP is specified on the LDR control statement, a listing is obtained describing where each module is loaded and what entry points and external symbols are used for loading. This listing is an overlay load map and is similar to the map of a nonoverlay load. A log of the directives used follows the map of the last overlay generated. If overlay loading aborts, the directives are not listed.

OBJECT LIBRARY MANAGEMENT 15

BUILD is an operating system utility program for generating and maintaining library datasets. A *library dataset* contains a program file followed by a directory file. Library datasets primarily provide the loader a means of rapidly locating and accessing program modules. The program file is composed of loader tables for one or more absolute or relocatable program modules. The directory file contains an entry for each program. The entry contains the name of the program module; the relative location of the program module in the dataset; and block names, entry names, and external names.

The BUILD program constructs a library from one or more input datasets named by the user when BUILD is called. A library dataset created by a BUILD run can be used as input to a subsequent BUILD run. Through BUILD directives, the user designates the program modules to be copied from the input datasets to the new library and their order in the library. However, no directives or control statement parameters are needed for the most frequent application of BUILD, which is to add new binaries from $BLD to an existing library of binary programs, replacing the old binaries where necessary.

BUILD does not use tape datasets.

BUILD CONTROL STATEMENT

Format:

BUILD,I=*idn*,L=*ldn*,OBL=*odn*,B=*bdn*,NBL=*ndn*,SORT,NODIR,REPLACE.

Parameters are in keyword form.

I=*idn* Name of dataset containing BUILD directives, if any. Directives can be included in the $IN dataset, or they can be submitted in a separate dataset.

If the I parameter appears alone or is omitted, all directives are taken from the $IN dataset, starting at its current position and stopping when an end-of-file is read.

If I=*ddn*, all directives are taken from the specified dataset, *ddn*, stopping when an end-of-file is read.

If I=0, no directives are read. The most common condition is to merge the modules from *odn* (the OBL dataset) with those from *bdn* (the B dataset), replacing OBL modules with B modules whenever the names conflict, and to write the output to *ndn* (the NBL dataset). Note that the input dataset specified by the B parameter corresponds to the binary output from CAL and CFT, also designated by B.

L=*ldn* Name of list output dataset.

If the L keyword appears alone or is omitted, list output is written to $OUT.

If L=*ldn*, list output is written to *ldn*.

If L=0, no list output is written.

OBL=*odn* Name of the first input dataset, usually a previously created library dataset.

If the OBL parameter is omitted or appears alone, the first dataset read is $OBL.

If OBL=*odn*, the first dataset read is *odn*.

If OBL=0, no old binary library exists. This is a creation run.

B=*bdn* Name of the second input dataset, whose modules will be added to or will replace the modules in the first dataset.

If the B parameter appears alone or is omitted, the second dataset read is $BLD.

If B=*bdn* is specified, the second dataset read is *bdn*, which is read to the first end-of-file.

If B=0, no modules are being added. This run edits an old library.

NBL=*ndn* Name of the output dataset, usually a new library dataset. If the NODIR parameter is also present, *ndn* is not in library format.

If the NBL parameter appears alone or is omitted, output is written to $NBL.

If NBL=*ndn*, output is written to *ndn*.

If NBL=0, no output is written.

SORT Specifies that all modules are to be listed alphabetically according to their new names. The default is to list the modules in the order they are first read. Note that SORT only applies to the list dataset and not to the output library.

NODIR Specifies that no directory is to be appended to the output dataset, resulting in an ordinary sequential dataset like $BLD. The default is to append the directory.

The dataset *ndn* specified by NBL is not rewound if NODIR is specified.

REPLACE Specifies that the output library is to contain modules in the same order as the old library. If omitted, the new library contains modules from the old library which are not replaced by modules from the input binary dataset, followed by modules from the input dataset, whether the module from the input dataset replace modules from the old library, or are new, in the order encountered on the input dataset.

Any of the following errors causes BUILD to abort:

- A module specified explicitly in a COPY or OMIT directive is not in the current input dataset.
- A module specified explicitly in a COPY directive has already been selected for output.
- Improper syntax is used in the BUILD control statement or in the directive dataset.
- An unrecognized directive or control statement keyword is used.
- A dataset name or module name is too long or contains illegal characters.

PROGRAM MODULE NAMES

BUILD directives refer to program modules by their names as given in the directory or, if the directory is missing or is unrecognizable, by the names given in the program modules.

PROGRAM MODULE GROUPS

In the COPY and OMIT directives, program modules with names containing one or more identical groups of characters can be specified together. To accomplish this, variable parts of each name are replaced by one or more hyphens. For example, XYZ- represents all names beginning with XYZ, including XYZ itself. In the extreme case, a name consisting of only a hyphen represents all possible names.

In addition, up to eight asterisks can be used anywhere in a name as wild characters matching any character other than a blank. For example, GE* specifies a group of modules having 3-character names including GET and GEM but not GE or GEMS, although GE*S could represent GEMS.

PROGRAM MODULE RANGES

In order to facilitate the copying of large numbers of contiguous program modules, the COPY directive allows use of a range specifier instead of a single name or group specifier. The range specifier has the general form:

(*first*,*last*)

which means: skip to the first module specified and copy all modules from the first up to and including the last module specified.

FILE OUTPUT SEQUENCE

If the SORT parameter appears in the BUILD control statement, all modules are copied alphabetically according to their new names. In the absence of a SORT parameter, modules are written in the order they are originally read from the input datasets.

The order of the entries in the directory is always the same as the order of the modules themselves.

FILE SEARCHING CONSIDERATIONS

The user need not be aware of the order of modules in the input dataset unless (1) two or more modules have the same name or (2) a range is specified in a COPY directive.

If two or more modules with the same name are in the input datasets, the last of the modules read is the one that survives, unless the user specifically omits that last module while its original dataset is the currently active input dataset.

The concept of *current position* in the input file is used to interpret range specifiers where the first name is omitted as in (,*last*) or (,). In such cases, the current position is defined to be either immediately after the last module copied or at the beginning of the dataset if no modules have yet been copied.

BUILD DIRECTIVES

BUILD is controlled through directives in a dataset defined by the I parameter on the BUILD control statement. A directive consists of a keyword and, if the keyword requires it, a list of dataset names or module names. When names are required, the keyword must be separated from the first name by a blank; subsequent names (if any) in the list are separated from each other by commas. Extra blanks are optional except within the keyword.

A line can contain more than one directive; periods or semicolons are used to separate directives on the same line from each other. A directive cannot be continued from one directive line to the next.

Examples of directives:

```
OMIT ENCODE,DECODE

COPY **CODE.
```

Examples of multiple directives on one line:

```
FROM OLDLIB; LIST; OMIT ENCODE,DECODE,XLATE

FROM $BLD. LIST.
```

FROM DIRECTIVE

A FROM directive names a single dataset, which is thus established as the input dataset for succeeding COPY, OMIT, and LIST directives, or it lists several datasets that (except for the last dataset in the list) are to be copied in their entirety to the output dataset ($NBL). The last dataset in the list is established as the current input dataset, just as if it were specified alone in the FROM directive. If no COPY or OMIT directive follows, the last dataset is also copied in its entirety to the output dataset.

An input dataset can be a library (with a directory) or an ordinary sequential dataset (such as $BLD). BUILD always determines whether a directory is present at the end of the dataset and attempts to use it if it is there. A library dataset is treated as sequential if its directory file is unrecognizable any reason.

Format:

FROM $dn_1,dn_2,\ldots,dn_n$

The following rule allows the user to copy several datasets with one FROM directive or to omit COPY (which means copy all) when it would be the only directive (except for OMIT directives) in the range of a particular FROM directive:

> If any dataset named on a FROM directive is not acted on by any LIST or COPY directive, then BUILD copies all of the modules belonging to that dataset. BUILD takes this action when it encounters the next FROM dataset name or the end of the directive file, whichever comes first.

If there are two input datasets to be read as soon as BUILD begins to execute (that is, if neither OBL=0 nor B=0 is specified), the modules from these two datasets are treated as if they belong to a single dataset as far as the OMIT, COPY, and LIST directives are concerned. However, if either of them is named in a FROM directive, it is treated as a separate dataset and OMIT, COPY, and LIST directives apply only to whichever is the current input dataset.

OMIT DIRECTIVE

The OMIT directive allows a user to specify certain modules otherwise included in a group be omitted from the group on subsequent copy operations. An OMIT affects modules on the current input dataset only; its effect ends when a FROM directive is encountered.

Format:

OMIT $fn_1, fn_2, \ldots, fn$

Each fn_i can be one of the following:

- A single name, such as $AB@CDEF or CAB22, by which binary records can be explicitly prevented from being copied, or

- A group name, such as F$- or *AB**, by which binary records are prevented from being copied unless they are specified explicitly (that is, singly) in a COPY directive (see the introduction to this chapter under Program Module Groups for a description of * and - usage).

If an *fn* parameter specifies a module not in the input dataset or a group of modules having no representatives in the input dataset, a diagnostic message is included in the list output and BUILD aborts.

COPY DIRECTIVE

COPY directives cause BUILD to select the specified modules for copying from current input dataset to the output dataset. The user specifies single modules, groups of modules, or ranges of modules to be copied. If the user specifies a module not in the current input dataset, a diagnostic message is included in the list output and BUILD aborts.

Format:

COPY $fn_1, fn_2, \ldots, fn_n$

Each fn_i is either of the two forms valid in OMIT directives:

- A single module name by which modules are explicitly selected for copying even if they belong to a group named in a previous OMIT directive, or

- A group specifier by which all the modules in the group are selected for copying unless they are specified either explicitly or implicitly in a previous OMIT directive.

In addition, two special forms are allowed for each fn_i in COPY directives:

- A form to rename a single module whose old name is specified explicitly; for example, OLDNAME=NEWNAME. (The name is changed both in the output directory and in the module's Program Description Table.)

- A form to copy an inclusive range, as in (FIRST,LAST), by which all the modules in the range are selected for copying unless they are specified either explicitly or implicitly in a previous OMIT directive.

These two forms are mutually exclusive. A module copied by being included in a range cannot at the same time be renamed. Nor can either form accept a hyphen or asterisk specifying a group of modules.

Examples:

BUG=ROACH	Copies BUG, renaming it to ROACH
(LOKI,THOR)	Copies all modules from LOKI through THOR
(THOTH,)	Copies all modules from THOTH to the end of the input dataset
(,ISIS)	Copies all modules from the current dataset position through ISIS
(,)	Copies all modules from the current dataset position to the end of the input dataset

The current dataset position is defined as the beginning of the input dataset if no modules have been selected for copying yet, or else as the beginning of the record immediately after the last module that has been selected for copying.

LIST DIRECTIVE

The LIST directive tells BUILD to list the characteristics of the modules in the current input dataset. Its effect is immediate. (BUILD's standard list output describes the contents of the output dataset and is produced at the end of the run so as not to interfere with output triggered by LIST directives.)

Format:

```
LIST
```

EXAMPLES

The following are examples of various uses of the BUILD program:

- Creating a new library dataset, using as input whatever binary modules have been written out to $BLD (for example, by CAL and/or CFT).

  Control statements:

  ```
  BUILD,OBL=0,I=0.
  SAVE,DN=$NBL,PDN=MYLIB.
        .
        .
  ```

- Adding one or more modules to an already existing library dataset, again taking the input from $BLD.

  Control statements:

  ```
  ACCESS,DN=$OBL,PDN=MYLIB.
  BUILD,I=0.
  SAVE,DN=$NBL,PDN=MYLIB.
        .
        .
  ```

  Any modules whose names were already in the directory of MYLIB are replaced by the new binaries from $BLD in the new edition of MYLIB that is created by BUILD and saved by the SAVE control statement.

- Merging several libraries.

  Control statements:

  ```
  ACCESS,DN=LIBONE,PDN=HERLIB.
  ACCESS,DN=LIBTWO,PDN=HISLIB.
  ACCESS,DN=ANOTHER,PDN=ITSLIB.
  ACCESS,DN=LASTONE,PDN=MYLIB.
  BUILD,I,OBL=0,B=0.
  SAVE,DN=$NBL,PDN=NEWLIB.
  ```

.
.

Directives:

```
FROM LIBTWO,ANOTHER,LIBONE,LASTONE
```

The order of the dataset names in the FROM directives, not the order of the ACCESS control statements, determines the order of processing. If two datasets contain modules of the same name, the surviving module is the one in the dataset whose name occurs later in the FROM directive. (Any module could be renamed before input from a succeeding dataset is begun, in order to prevent it from being discarded. Note the section on File Searching Considerations in the introduction to this chapter for a description of the interaction with OMIT directives.)

- Deleting a program module from a library.

  Control statements:

  ```
  ACCESS,DN=$OBL,PDN=MYLIB.
  BUILD,B=0.
  SAVE,DN=$NBL,PDN=MYLIB.
        .
        .
        .
  ```

  Directive:

  ```
  OMIT BADPROG
  ```

- Extracting a program module from a library for input to the system loader, using the local dataset name $BLD as the intermediate file.

  Control statements:

  ```
  ACCESS,DN=XXX,PDN=MYLIB.
  BUILD,I,OBL=XXX,B=0,NBL=$BLD,NODIR.
        .
        .
        .
  ```

  Directive:

  ```
  COPY RUNPROG
  ```

JOB CONTROL LANGUAGE STRUCTURES 16

This section discusses three aspects of job control language structures:

- Control statements
- Job control language expressions
- Procedures

CONTROL STATEMENTS

The COS job control language allows three fundamental logic structures:

- *Simple control statement sequences.* Control statements are processed one after another.
- *Conditional control statement blocks.* A sequence of control statements is processed only if the specified condition is met.
- *Iterative control statement blocks.* A sequence of control statements is processed repetitively until the specified condition is met.

Most computer algorithms can be expressed in terms of the three above structures or as combinations of them.

SIMPLE CONTROL STATEMENT SEQUENCES

A simple control statement sequence is a series of one or more of the control statements described in sections 6 through 15 of this manual. The individual control statements are processed sequentially as described in section 3 of this manual.

CONDITIONAL CONTROL STATEMENT BLOCKS

A conditional control statement block is a group of control statements that is processed only if a specified condition is met. The control statements IF, ELSE, ELSEIF, EXITIF, and ENDIF allow other control statements to be placed in a conditional block structure.

- IF defines the beginning of a conditional block.
- ENDIF defines the end of a conditional block.
- ELSE is used to define an alternate condition.
- ELSEIF defines an alternate condition to test when the previous one tested is false.
- EXITIF defines a condition which causes an escape from a conditional block.

ELSE, ELSEIF, and EXITIF sequences are optional.

IF - Begin conditional block

The IF control statement defines the beginning of a conditional block. Each IF control statement must have a corresponding ENDIF control statement. IF is a system verb.

Format:

```
IF(expression)
```

Parameter:

expression
A valid JCL expression (discussed later in this section). This parameter is required.

ENDIF - End conditional block

The ENDIF control statement defines the end of a conditional block. ENDIF is a system verb.

Format:

```
ENDIF.
```

Parameters: None

<u>ELSE - Define alternate condition</u>

The ELSE control statement is used to define an alternate condition. An IF statement, as well as any ELSEIF statements, must precede the ELSE control statement. If all conditions specified by the IF and ELSEIF statements that precede the ELSE in the conditional block test as false, then the sequence of statements that follow the ELSE statement is executed.

Within a conditional block, only one ELSE sequence is permitted. The ELSE statement, if present, must follow any ELSEIF statement.

ELSE is a system verb.

Format:

```
ELSE.
```

Parameters: None

<u>ELSEIF - Define alternate condition</u>

The ELSEIF control statement defines an alternate condition to test if the previously tested condition was false. The sequence of statements following the ELSEIF statement is executed when the ELSEIF expression is true. All ELSEIF control statements must precede the optional ELSE control statement for a conditional block. An ELSEIF statement without a previously processed IF statement results in a job step abort. An unlimited number of ELSEIF sequences can be used in a conditional block.

ELSEIF is a system verb.

Format:

ELSEIF(*expression*)

Parameter:

expression
: A valid JCL expression (discussed later in this section). This parameter is required.

A conditional block can contain any number of ELSEIF control statements. The block of control statements following an ELSEIF statement is processed under the following conditions:

- The expression for the IF statement is false.
- All preceding ELSEIF statement expressions are false.
- The ELSEIF expression is true.

EXITIF - Exit from conditional block

The EXITIF control statement defines the conditions that must be met so that the remaining control statements in the conditional block are skipped. EXITIF is a means of skipping to the ENDIF statement without regard to EXIT statements. If the EXITIF expression is true, the remainder of the conditional block is skipped; if the expression is false, the control statements which follow the EXITIF statement are executed.

EXITIF may appear anywhere within a conditional block. An EXITIF statement that is not within a conditional block causes a job step abort. When conditional blocks are nested, the EXITIF control statement applies to the innermost conditional block which contains it.

EXITIF is a system verb.

Formats:

```
EXITIF.

EXITIF(expression)
```

Parameter:

expression
: A valid JCL expression (discussed later in this section). If *expression* is omitted, the remainder of the block is skipped unconditionally.

Conditional block structures

The conditional block is first scanned to verify the validity of the block's syntax. If any syntax errors exist, the block is skipped without being evaluated and a job step abort error occurs. Note that any EXIT control statements within the conditional block are ignored when a syntax error exists in that conditional block. This validation occurs when the control statement file, where it is contained, is invoked. (Validation occurs at job initiation if the control statement file is $CS; it can also occur at the time that a procedure is invoked, or when a CALL statement is encountered.)

Null sequences (for example, an ELSE statement immediately following an ELSEIF) are ignored without comment.

Conditional blocks can be constructed in the following ways:

- Basic conditional block
- Conditional block with ELSE
- Conditional block with ELSEIFs
- Conditional block with ELSEIFs and ELSE

Basic conditional block - The format of a basic conditional block (figure 16-1) begins with an IF statement and ends with an ENDIF statement. When the IF statement expression is true, the control statement sequence that follows is processed. If the expression is false, the control statement sequence is not processed.

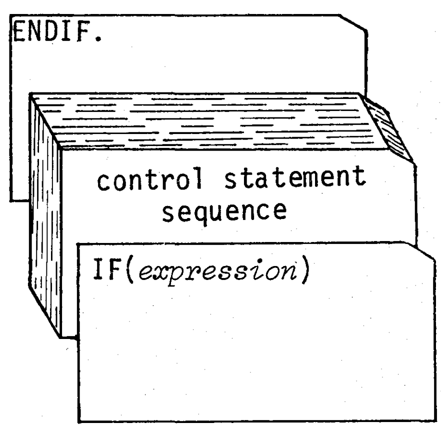

Figure 16-1. Basic conditional block structure

Example:

Following is an example of the conditional block structure.

```
ACCESS,DN=MYPROG.
MYPROG.
EXIT.
IF(PDMST.EQ.1)
  *.
  *.      UNEXPECTED JOB STEP ABORT ERROR
  *.
  EXIT.
ENDIF.
```

In this example, if the ACCESS request or execution of MYPROG fails, the conditional block after the EXIT control statement is processed. The conditional block determines if the job step abort occurred because the ACCESS (for example, the dataset was not found), in which case the processing of control statements resumes after the ENDIF control statement. If this is not the reason for the abort, the job terminates with the EXIT control statement.

Following is an example of a conditional block using EXITIF.

```
ACCESS,DN=MYPROG,NA.
IF(PDMST.NE.1)
           UPDATE(Q=MYPROG)
           CFT(I=$CPL,ON=A)
           LDR(AB=MYPROG,NX,USA)
           SAVE(DN=MYPROG,NA)
           EXITIF.
           EXIT.
*.
*.          ERROR GENERATING MYPROG
*.
          EXIT.
ENDIF.
MYPROG.
```

In this example, a conditional block is used to generate a dataset if that dataset is not found. EXITIF is used to skip the remaining statements in the conditional block if the dataset is generated successfully. Otherwise, the job terminates.

Conditional block with ELSE - The second conditional block structure includes the ELSE control statement. The control statement sequence is processed if the expression on the IF statement is true. If the expression is not true, the sequence following the ELSE statement is processed. The block structure is illustrated in figure 16-2.

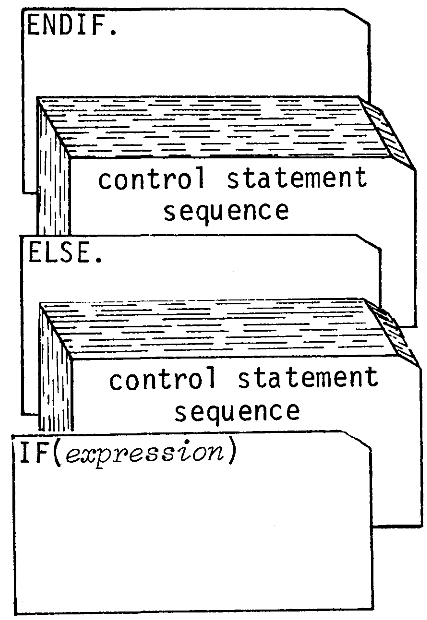

Figure 16-2. Conditional block structure including ELSE

Example:

An example of a conditional block structure using the ELSE statement follows.

```
ACCESS,DN=INITJCL.
ACCESS,DN=PREPROG.
ACCESS,DN=PROG.
PREPROG.
IF(JSR.NE.0)
  CALL,DN=INITJCL.
  SWITCH,1=ON.
ELSE.
  SWITCH,1=OFF.
ENDIF.
PROG.
```

After PREPROG is executed, the conditional block determines if PREPROG has successfully executed (by its setting of JSR). The procedure INITJCL is executed and a sense switch is set if JSR is nonzero. The sense switch is cleared if PREPROG set JSR to zero.

Conditional block with ELSEIF - The third conditional block structure (figure 16-3) includes one or more ELSEIF statements. Each logical expression on the IF and ELSEIF statements is tested in sequence until a true condition is found; then the corresponding control statement sequence is processed.

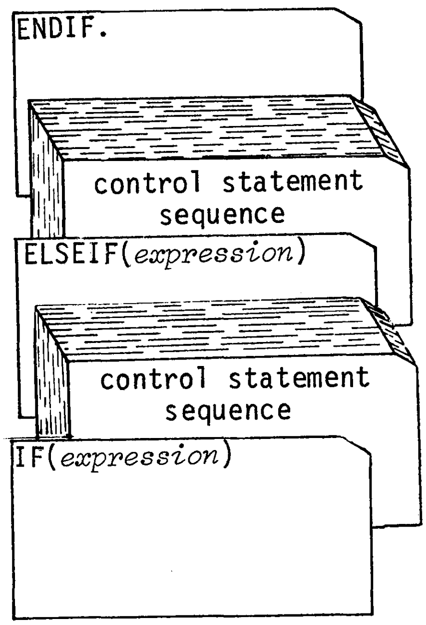

Figure 16-3. Conditional block structure including ELSEIF

A conditional block can contain any number of ELSEIF control statements. The block of control statements following an ELSEIF statement is processed under the following conditions:

- The expression for the IF statement is false.
- All preceding ELSEIF statement expressions are false.
- The ELSEIF expression is true.

Example:

An example of a deck including the ELSEIF statement is:

```
IF(SYSID.EQ.'COS 1.07')
  ACCESS,DN=$FTLIB,ID=V107.
ELSEIF(SYSID.EQ.'COS 1.08')
  ACCESS,DN=$FTLIB,ID=V108.
ELSEIF(SYSID.EQ.'COS 1.09')
  ACCESS,DN=$FTLIB,ID=V109.
ENDIF.
LDR,NOLIB,LIB=$FTLIB.
```

This conditional block tries to access the correct version of the FORTRAN library, $FTLIB, for the execution of the loader following the conditional block.

Conditional block with ELSEIF and ELSE - The fourth conditional block structure, shown in figure 16-4, uses ELSEIF and the ELSE statements. A block can contain any number of ELSEIF statements but can contain only one ELSE, which must be the last conditional statement before the ENDIF.

The ELSE control statement sequence in this case is processed only if:

- The expression on the IF statement is false, and
- All ELSEIF statement expressions are also false.

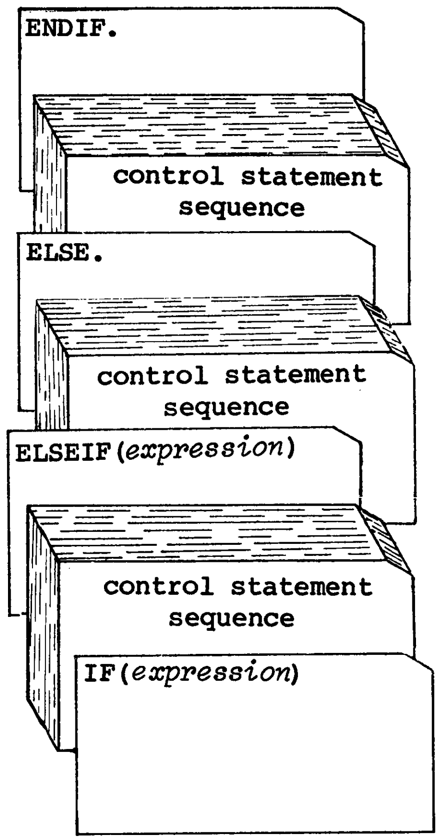

Figure 16-4. Conditional block structure including ELSEIF and ELSE

Example:

This example is an expansion of the example for the third format and allows execution of the compiled program if there is enough time left and if the correct library is accessible. On a successful run, the dataset called RESULTS is disposed as a staged dataset.

```
IF(TIMELEFT.GT.175)
  IF(SYSID.EQ.'COS 1.08')
    ACCESS,DN=$FTLIB,ID=V108.
  ELSEIF(SYSID.EQ.'COS 1.09')
    ACCESS,DN=$FTLIB,ID=V109.
  ELSE.
    *.
    *.  CURRENT SYSTEM LEVEL NOT RECENT ENOUGH
    *.
    EXIT.
  ENDIF.
  LDR,NOLIB,LIB=$FTLIB.
  SET,J1='YES'L.
ELSE.
  SET,J1='NOTIME'L.
ENDIF.
IF(J1.EQ.'YES'L)
  DISPOSE,DN=RESULTS,DC=ST.
ELSE.
  *.
  *.  JOB DID NOT RUN TO NORMAL COMPLETION
ENDIF.
EXIT.
```

ITERATIVE CONTROL STATEMENT BLOCKS

An iterative control statement block is the third fundamental logic structure allowed by the COS job control language. It contains a control statement sequence that is to be processed more than once during the processing of a job.

- LOOP defines the beginning of an iterative block.
- ENDLOOP defines the end of an iterative control statement block.
- EXITLOOP defines the conditions under which the control statement block iteration is to end.

LOOP - Begin iterative block

The LOOP control statement is required to define the beginning of an iterative block. An ENDLOOP control statement is required at the same nesting level to terminate the iterative block. LOOP is a system verb.

Format:

```
LOOP.
```

Parameters: None

ENDLOOP - End iterative block

The ENDLOOP control statement terminates an iterative control statement block. If an ENDLOOP control statement occurs without a preceding LOOP statement at the same nesting level, a job step abort occurs. Execution of the ENDLOOP statement results in control being passed to the preceding LOOP statement which begins another iteration of the loop.

Format:

```
ENDLOOP.
```

Parameters: None

EXITLOOP - End iteration

The EXITLOOP control statement defines the conditions under which the control statement block iteration is to end. If its expression is true, the loop is exited; if it is false, the control statements which follow are executed.

An EXITLOOP statement that does not appear within an iterative block causes a job step abort. When nesting iterative control statement blocks, the EXITLOOP control statement defines the exit conditions for only the most immediate iterative block. EXITLOOP is a system verb.

Formats:

```
EXITLOOP.

EXITLOOP(expression)
```

Parameter:

expression

Optional valid JCL expression (discussed later in this section). If omitted, an unconditional exit from the iterative block occurs.

Figure 16-5 illustrates an iterative control statement block.

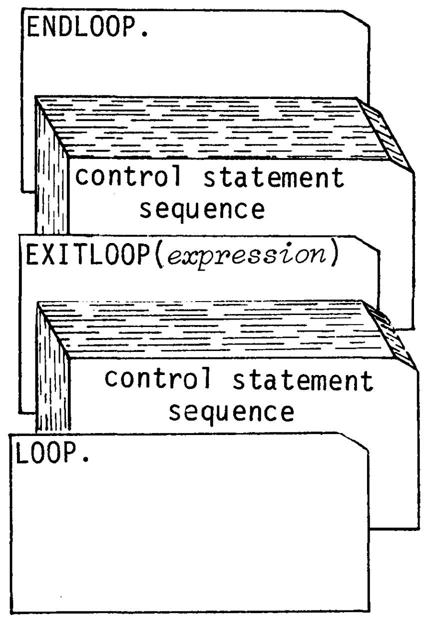

Figure 16-5. Iterative block structure

Iterative blocks are prescanned for syntax errors before actual processing begins. Any errors in the block structure cause a skipping of that block followed by a job step abort. If an iterative block is included within a conditional block, it must be totally contained within that conditional block.

Example:

The following example merges the two datasets DSIN1 and DSIN2 for 60 records.

```
SET,J1=0.
SET,J2=60.
LOOP.
  EXITLOOP(J2.EQ.0)
  IF(J1.EQ.0)
          COPYR,I=DSIN1,O=OUTDS.
          SET,J1=1.
```

```
   ELSE.
    COPYR,I=DSIN2,O=OUTDS.
    SET,J1=0.
   ENDIF.
   SET,J2=J2-1.
ENDLOOP.
REWIND,DN=DSIN1:DSIN2:OUTDS.
```

JOB CONTROL LANGUAGE EXPRESSIONS

Much of the power of the control statements described in this section derives from the use of *expressions*. Expressions allow operations such as incrementing counters, checking error codes, and comparing strings.

An *expression* is a *string* consisting of *operands* and *operators*. Expressions are evaluated from left to right, honoring nested parentheses and operator hierarchy. This subsection begins by defining operands and operators, and ends by discussing expression evaluation and strings.

OPERANDS

Expression *operands* are of four types:

- Integer constants
- Literal constants
- Symbolic variables
- Subexpressions

Integer constants

An *integer constant* is a character string with two possible forms:

$\pm$ *ddd*...

nnn...B

d is a decimal digit and *n* is an octal digit.

An integer constant has an approximate decimal range $0 \leq |I| \leq 10^{19}$. Range overflow is not detected and overflow results may be unpredictable.

Literal constants

A *literal constant* is a string of one to eight characters of the form:

'*ccc*...'L
'*ccc*...'R
'*ccc*...'H

c is a character code with an ordinal number in the the range 040 (octal) through 176. The value of a character constant corresponds to the ASCII character codes positioned within a 64-bit word. Alignment is indicated by the following suffixes:

L Left-adjusted, zero-filled
R Right-adjusted, zero-filled
H Left-adjusted, space-filled

If no suffix is supplied, H is assumed.

Symbolic variables

A *symbolic variable* is a string of one to eight alphanumeric characters, beginning with an alphabetic character.

A symbolic variable always has an associated value. COS defines a set of symbols when the job is initiated. Symbols are mnemonics for values maintained by COS and/or the user. The user can manipulate the group of symbols listed in table 16-1 through COS control statements or through system requests.

Certain symbols allow communication between COS and the job being processed. Used in the JCL block control statements defined in this section, these symbols provide the user with powerful tools for analyzing the progress of a job. For example, a job can request the reason for an abort situation and proceed, based on the reply from COS, through the use of conditional control statements.

Symbols that are preserved over subprocedure calls are called *local* to a procedure; they are saved when a subprocedure is called. Those that are not preserved are *global* over all procedures and can be altered by any procedure. *Constants* are symbols that are never altered.

Information on predefined symbols is summarized in table 16-1. In table 16-1, the only local symbols are J0 through J7.

Table 16-1. Symbolic variable table

Symbol	Set by	Range	Description
J0-J7	U	Any 64-bit value	Job pseudo-registers; represent user-alterable data local to a procedure. Each procedure level can be considered to have its own set of J registers.
G0-G7	U	Any 64-bit value	Global job pseudo-registers; represent user-alterable data global over all procedure levels. Data can be passed into or returned from procedures with the G registers.
JSR	U	Any 64-bit value	Job status register; previous job step completion code (normally 0).
FL	S	$0\text{-}77777777_8$	Current job field length; can be set with MEMORY statement.
FLM	S	$0\text{-}77777777_8$	Maximum job field length; determined by JOB statement.
SYSID	I	Literal value	COS system level of the form 'COS X.XX'
SID	I	Literal value	Mainframe identifier for front-end of job origin; 2 right-justified ASCII characters.
SN	I	64-bit integer	CPU serial number
SSW*n*	S	$(1 \le n \le 6)$	Job pseudo sense switch settings; can be set with the SWITCH statement.
ABTCODE	S	System error codes (See Appendix D) 0-*nnn*	COS job abort code; abort code corresponding to the last job step abort. The abort code corresponds to the abort message number (the *nnn* in AB*nnn*) issued by COS.
TRUE	I	-1	True value

U User
S COS
I System constant

Table 16-1. Symbolic variable table (continued)

Symbol	Set by	Range	Description
FALSE	I	0	False value
TIME *hh:mm:ss*	S	Literal value	Time of day in the form:
DATE	S	Literal value	Date in the form: *mm/dd/yy*
TIMELEFT	S	64-bit integer	Job time remaining in milliseconds as an integer value
PDMFC	S	64-bit value	Most recent user-issued Permanent Dataset Manager request. See Appendix D.
PDMST	S	64-bit value	Status of most recent Permanent Dataset Manager request. See Appendix D.

U User
S COS
I System constant

Subexpressions

A *subexpression* is an expression that is evaluated so that its result becomes an operand.

OPERATORS

Expression *operators* are of three types:

- Arithmetic
- Relational
- Logical

These operators are used in the FORTRAN sense. The expression operators are detailed in table 16-2.

Table 16-2. Expression operator table

Type	Function	Symbol	Results
A	Addition	+	64-bit sum of operands
A	Unary plus	+	Following integer operand is positive.
A	Subtraction	-	64-bit difference of operands
A	Unary minus	-	Following integer operand is negative.
A	Multiplication	*	64-bit product of operands
A	Division	/	64-bit quotient of operands
R	Equal	.EQ.	True/false
R	Not equal	.NE.	True/false
R	Less than	.LT.	True/false
R	Greater than	.GT.	True/false
R	Less than or equal	.LE.	True/false
R	Greater than or equal	.GE.	True/false
L	Inclusive OR	.OR.	A 1 bit in either operand sets corresponding bit in the result.
L	Intersection	.AND.	A 1 bit in both operands sets corresponding bit in the result.
L	Exclusive OR	.XOR.	A 1 bit is set in the result if either (but not both) corresponding bit in the operands is 1.
L	Unary complement	.NOT.	A 1 bit (or 0) is set in the result if the corresponding operand bit is 0 (or 1).

A Arithmetic
R Relational
L Logical

Arithmetic operators

All *arithmetic operations* are performed on 64-bit integer quantities. Care must be used with arithmetic operators because:

- Multiplication/division underflow or overflow of the result is not detected,
- Division by zero produces a zero result, and
- Intermediate and final results are truncated. For example, 2*(13/2) yields 12 whereas (2*13)/2 yields 13.

Relational operators

Relational operations return a -1 value for a TRUE result and a 0 value for a false result. A value produced by an arithmetic or logical operation is considered true if it is a negative value.

Logical operators

Logical operations return a 64-bit result. Their functions are performed on a bit-by-bit basis.

EXPRESSION EVALUATION

Expressions are evaluated from left to right, honoring nested parentheses. The operator hierarchy is:

1. Multiplication and division
2. Addition, subtraction, and negation
3. Relational operation
4. Complement (.NOT.)
5. Intersection (.AND.)
6. Inclusive OR (.OR.)
7. Exclusive OR (.XOR.)

Parentheses can be used to change the order of evaluation. For example, 2+3*4 is evaluated as 14 whereas (2+3)*4 is evaluated as 20.

****

CAUTION

Because COS does not check for type, the results of expression evaluation may not be as expected. For example, although both J1.EQ.1 and J2.EQ.2 are TRUE, (J1 .AND. J2) is FALSE.

****

STRINGS

A *string* is a group of characters which is to be taken literally as a parameter value.

- Strings are normally delimited with apostrophes, in which case they are referred to as *literal strings*.
- Strings can also be delimited with open and close parentheses, in which case they are referred to as *parenthetic strings*.

Characters in a string can be any ASCII graphic characters (codes 040_8 through 176_8).

Literal strings

Apostrophes are never treated as part of a literal string during evaluation except when doubled or when the literal string is a part of an expression (see examples). To continue literal strings across card images, place an apostrophe followed by a continuation character at the end of the line, and place the remainder of the string on the next card image preceded by an apostrophe. Characters otherwise recognized as separators are not evaluated as such when part of a literal string. Doubled apostrophes within a literal string are interpreted as a single apostrophe. A literal string without characters is the null string.

Examples:

String	Interpretation
...'LITERAL STRING'	LITERAL STRING
...'LITERAL STRING'^ 'ACROSS CARD IMAGES'	LITERAL STRINGACROSS CARD IMAGES

String	Interpretation
...'WON''T SHOW'	WON'T SHOW
...'LITERAL ^ STRING'	LITERAL ^ STRING
...''	Null string
...IF (G0.EQ.'COS1.01'L)	G0.EQ.'COS1.01'L

Parenthetic strings

There are two main differences between parenthetic strings and literal strings: in parenthetic strings, (1) blanks are removed, and (2) some separators are evaluated. The separators are evaluated as follows:

- If apostrophes appear in a parenthetic string, the enclosed characters are interpreted as a literal string.
- The continuation character is interpreted within a parenthetic string.
- Nested parentheses within a parenthetic string are not treated as separators.

Examples:

String	Interpretation
...(LITERAL STRING)	LITERALSTRING
...(LITERAL STRING ACROSS CARD IMAGES)	LITERALSTRINGACROSSCARDIMAGES
...(WON''T SHOW)	WONTSHOW
...((NESTED PARENTHESES))	(NESTEDPARENTHESES)
...(STRING 'LITERAL STRING')	STRINGLITERAL STRING
...(CLOSED PARENTHESIS ')')	CLOSEDPARENTHESIS)
...(KEYWORD=ABC.DEF)	KEYWORD=ABC.DEF
...( )	Null string
...()	Null string

PROCEDURES

Just as FORTRAN programs can be divided into separate modules called subprograms, control statement sequences can be divided into modules called *procedures*. A *procedure* is a sequence of control statements or data or both that have been saved for processing at a later time. Procedures simplify control statement use in three ways:

- Generalized procedures can be written to perform many similar tasks. Work is saved because a new control statement sequence need not be written to perform each separate task.
- Complex control statement structures can be decomposed into separate subtasks, with a separate procedure written for each subtask. Such modularization reduces the job's design complexity and allows each subtask to be individually tested.
- Procedure libraries can be built. Procedures need be defined only once and placed in a library; different jobs and users can use the procedures and make them part of their own control statement structures.

Procedures have two formats:

- A *simple procedure* consists of only the control statement body.
- A *complex procedure* consists of a prototype definition statement, control statement body, and optional data.

SIMPLE PROCEDURES

A simple procedure is a series of control statements that does not reside in the primary control statement dataset ($CS). No parameter substitution occurs in a simple procedure.

Since a simple procedure has no name associated with it, a simple procedure can only reside in a nonlibrary dataset. It therefore, must be invoked with the CALL control statement without the CNS parameter.

Example:

The first file of dataset MOVER contains five control statements. The five control statements can be executed with the following procedure calling statement:

```
CALL,DN=MOVER.
```

In the above example, interpretation of control statements from dataset MOVER terminates when a RETURN statement is encountered (see section 7 of this manual), when the end of the first file (in dataset MOVER) is reached, or an EXIT statement.

COMPLEX PROCEDURES

A complex procedure provides the capability of replacing values within the procedure body with values supplied from the procedure call. These values are called *substitution parameters* and are governed by the prototype statement of the procedure.

A complex procedure can reside in a library or nonlibrary dataset.

Complex procedures are invoked (executed) in one of two fashions:

- Procedure name call. The procedure must first reside in a known control statement library (either $PROC or a local dataset named with a LIBRARY control statement); the procedure is called (invoked) by using the procedure name as the control statement verb.

- CALL statement call. The procedure must reside in the first file of a separate dataset; the dataset is named in the CALL control statement. The CNS (crack next statement) parameter must be used for the operating system to properly recognize and process the procedure prototype statement. PROC and ENDPROC are not used with CALL.

Complex procedures can be defined within the control statement stream (*in-line definition*) or as input to the BUILD utility.† When an in-line procedure definition is encountered in the JCL control statement file, it is processed and written to the system default library $PROC. See example 8 later in this section for an example of how to create a user permanent procedure library.

A complex procedure can contain *formal parameters* that define what substitution is to occur in the procedure body. A character string that is eligible for substitution is listed in the prototype statement as a *formal parameter specification*. This name, when preceded by an ampersand in the definition body, indicates that a value is to be substituted during procedure invocation. COS replaces the ampersand and parameter name with corresponding value supplied by the procedure invocation. If the parameter listed in the prototype statement is not preceded by an ampersand in the body, substitution does not occur. If two ampersands precede the string, one is removed and substitution is inhibited.

---

† BUILD currently does not suppport procedure entries in libraries.

Any string consisting of one through eight characters (ampersand included) can be selected for substitution.

When a statement in the current control statement file calls a procedure, COS searches the definition body for the character strings preceded by ampersands. For each occurrence, COS substitutes the values supplied by either the calling statement or the prototype statement.

Whereas simple procedures consist only of a control statement body, complex procedures contain five elements as shown in figure 16-6.

- PROC defines the beginning of an in-line procedure definition block.

- The prototype statement specifies the name of the procedure and identifies character strings within the procedure that are to be substituted when the procedure is called. COS uses values supplied with the procedure call and default parameter values from the prototype statement to replace these strings.

- The procedure definition body is a sequence of COS control statements processed as part of the current control statement file when the procedure is called. It can optionally include lines of text data preceded in the definition body by an &DATA control statement.

- &DATA introduces text information to be included in the procedure definition body, and names the dataset to be created and written to when the procedure is invoked. When the procedure is invoked, the named dataset is created and the text information is available in that local dataset, including any substitutions resulting from the call. This temporary dataset remains local and allows programs such as CAL or CFT to use the temporary dataset as source data.

- ENDPROC indicates the end of an in-line procedure definition block.

The first control statement in an in-line procedure is PROC; the last is ENDPROC. A prototype statement follows PROC providing the name of the procedure and optionally a list of parameters that identify the substitution values within the definition body.

In addition to defining the values to be substituted, the prototype statement parameters control the selection or omission of the parameters and define the default value assignments. The control statements and data to be processed are contained in the definition body. The control statements are grouped in a sequence.

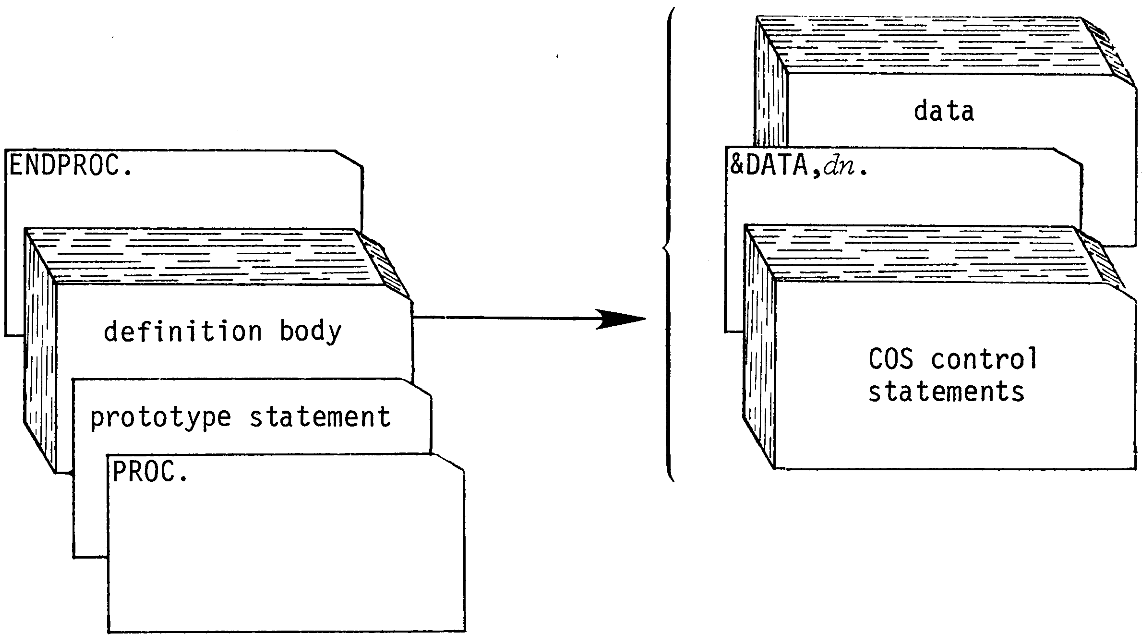

Figure 16-6. Procedure definition deck structure

If data is included in a procedure, the data is preceded by an &DATA statement and follows the control statement sequence. The &DATA statement also includes the name of the dataset to which the data is to be written after processing so that programs can use the data as source data.

A definition can be placed within a definition; such nesting can occur to any level. However, nested definitions do not become defined until the outermost procedure is invoked.

PROC - Begin procedure definition

The PROC control statement defines the beginning of an in-line procedure definition block. PROC is a system verb.

Format:

```
PROC.
```

Parameters: None

Prototype statement - Introduce a procedure

The prototype control statement has two functions: (1) to specify the name of the procedure and (2) to provide the *formal parameter specifications* that define where substitution is to occur within the definition body. Value substitution is described later in this section.

Format:

$name,p_1,p_2,p_3,\ldots,p_n.$

Parameters:

name Procedure name; 1 to 8 alphanumeric characters. The name should not be the same as a system verb; if it is, the results are unpredictable.

p_i Formal parameter specifications, using one of the formats listed below. A formal parameter identifies a character string within the definition body. All formal positional parameters, if any, must precede all formal keyword parameters; if they do not, the procedure definition is in error and the job aborts.

pos_i Positional formal parameter specification

key_i= *dvalue*:*kvalue*
Keyword formal parameter specification as follows:

key_i Formal keyword parameter

dvalue Optional default value; this value is substituted if entire keyword parameter is omitted from the calling statement.

kvalue Optional keyed default value; this value is substituted if the keyword is present but no value is specified.

Special cases:

key_i= Provides no default values and requires the caller to provide a non-null value.

key_i=: Provides no default values, but allows the user to specify key_i= or just key_i.

PROCEDURE DEFINITION BODY

The procedure definition body consists of a sequence of COS control statements processed as part of the current control statement file when the procedure is called. (It can optionally include lines of text data preceded in the definition body by an &DATA control statement. See &DATA, which follows.)

The prototype statement identifies character strings within the procedure that are to be substituted when the procedure is called. COS uses values supplied with the procedure call and default parameter values from the prototype statement to replace these strings.

An ampersand (&) must precede each parameter to be substituted (*substitution parameter*) within the definition body. If a parameter appears in the prototype, a matching string in the body is found but not preceded by an ampersand, substitution does not occur.

&DATA - Procedure data

Data can be included within the procedure definition body after the procedure data card.

The *dn* parameter creates a temporary dataset composed of the data identified in the procedure, including any substitutions resulting from the call. This temporary dataset allows programs such as CAL or CFT to use it as source data.

Format:

```
&DATA,dn.
```

Parameter:

dn Name of dataset to contain the data that follows; *dn* is required.

The initial separator for an &DATA statement can be a blank, comma, or an open parenthesis; the statement terminator can be a blank, period, or a close parenthesis.

An &DATA specification cannot be continued to subsequent cards. All card images following an &DATA card up to the next &DATA card are written to the specified dataset after string substitution is performed. See example 7 later in this section.

ENDPROC - End procedure definition

The ENDPROC control statement indicates the end of an in-line procedure definition block. ENDPROC is a system verb.

Format:

```
ENDPROC.
```

Parameters: None

PARAMETER SUBSTITUTION

Formal parameter specifications can be selected for substitution. Character strings to be substituted are delimited by any character other than numerals, alphabetics, commercial at (@), dollar sign ($), and the percent sign (%). An ASCII underline is used as a string delimiter when the next character is one of these characters. See example 3 later in this section. COS deletes the underline after evaluating the string it delimits. Thus, the underline concatenates the strings it delimits.

Formal parameter specifications can be in positional or keyword format.

Positional parameters

Positional formal parameters allow the user to list the strings within the body that can be substituted. The calling statement lists values to be substituted for these strings in the same order in which they are listed in the prototype statement. The value supplied with the calling statement is substituted for every occurrence of the corresponding formal positional parameter within the definition body. If the caller passes too few positional parameters, null strings are substituted for the remaining formal positional parameters. If too many positional parameters are passed, the procedure call is in error and the job aborts.

Keyword parameters

Keyword formal parameters are listed in any order after all positional parameters are given on the prototype statement and the calling statement. A keyword formal parameter allows the user to specify substitution values on the prototype statement that are to be used when one is not given on the calling statement.

If the keyword formal parameter is included in the calling statement with a value, that value is substituted. If the entire keyword formal parameter is omitted from the calling statement, the *default value* on the prototype statement is substituted. If a default value is not provided on the prototype statement, the character string within the body corresponding to that formal parameter is not included in the procedure expansion.

If only the keyword portion of the keyword formal parameter (the character string itself) is included in the calling statement, without a value assigned to it, then a *keyed default value* from the prototype statement is substituted. If a keyed default value is not provided on the prototype statement, again the character string within the body corresponding to that formal parameter is not included in the procedure expansion.

A keyword parameter enclosed in apostrophes ('*KEY=value*') is considered a positional parameter.

The forms of keyword substitution are summarized in table 16-3.

Positional and keyword parameters

When supplying both positional and keyword parameters, all positional parameters must precede all keyword parameters; COS evaluates the call's positional parameters first. The end of the caller's list of positional parameters is signaled by the appearance of a keyword parameter, statement terminator, or by specifying all positionals.

Table 16-3. Keyword substitution after expansion

Calling Statement \ Keyword Format for Prototype Statement	*key* (positional)	*key=:kv* *key=:(kv)*	*key=dv:kv* *key=(dv):kv* *key=dv:(kv)*	*key* (not positional)	*key=dv* *key=(dv)*
1. *name, value.*	*Value*	CS119	CS119	CS119 and CS122	CS 119
2. *name, key.*	CS121	*kv*	*kv*	CS121	CS121
3. *name.* *name,* ().	Null	Null	*dv*	CS122	*dv*
4. *name, key=value.*	*Value*	*Value*	*Value*	*Value*	*Value*
5. *name, key=.*	Null	Null	Null	Null	Null

kv=keyword value
dv=default value

Error messages:

CS119 - EXTRA POSITIONAL PARAMETER: *value*
CS121 - KEYWORD USED WITHOUT ASSIGNING IT A VALUE: *key*
CS122 - NO VALUE WAS ASSIGNED TO *key*

Apostrophes and parentheses

Sometimes parameter values in a procedure definition or a procedure calling statement require a special format. If a literal string (a string delimited with apostrophes) appears in either of these statements, it is processed as if it were a literal constant. That is, all apostrophes in the value remain when the value is substituted. See example 5 later in this section.

To avoid any possibility of erroneous processing, use parentheses as string delimiters in these statements. Outermost parentheses preceded by the initial, parameter, equivalence, or concatenation separators are removed during value substitution. This procedure delays processing of any separator characters in the string until the statement itself, with substituted values, is processed.

This delay is also required when specifying multiple values for the default value and/or keyed default value parameters on a procedure definition statement. See examples 1, 2, 4, and 6. Parentheses are advised in the procedure calling statement when the use of the value in the procedure statements is unknown. See examples 4, 5, and 6 later in this section.

The forms of parenthetical substitution are summarized in table 16-4.

Table 16-4. Expansion of parenthetic and literal string values

Invocation	Expansion
value	*value*
(*value*1=*value*2)	*value*1=*value*2
*value*1'.'*value*2	*value*1'.'*value*2
*value*1(.)*value*2	*value*1.*value*2

Examples:

The following examples demonstrate the COS control statement procedure substitution process.

Example 1:

Consider a single statement procedure called LOAD defined as follows:

Definition

```
PROC.
LOAD,NOGO=:NX,LIBRARY=($FTLIB:$SYSLIB):MYLIB.  Prototype statement
LDR,&NOGO,LIB=&LIBRARY.                        Definition body
ENDPROC.
```

The prototype statement in this example defines two formal parameters, both of which are in keyword format. The keyword NOGO has a null value when omitted from the calling statement and a value of NX when included on the calling statement in keyword-only format. The keyword LIBRARY has the default value of $FTLIB:$SYSLIB. When LIBRARY is used in the calling statement without a value, the keyed default value, MYLIB, is substituted.

When the LOAD procedure is invoked, it expands to a single statement whose form depends on the choice of parameters:

Invocation

```
LOAD,NOGO.
LOAD.
LOAD,LIBRARY=THISLIB.
LOAD,LIBRARY,NOGO.
```

Expansion

```
LDR,NX,LIB=$FTLIB:$SYSLIB.
LDR,,LIB=$FTLIB:$SYSLIB.
LDR,,LIB=THISLIB.
LDR,NX,LIB=MYLIB.
```

Example 2:

The following in-line procedure definition creates a procedure called BLDABS.

Definition

```
PROC.
BLDABS,SOURCE,LIST,GO='NO':'YES',LIB=
   :($SYSLIB:$FTLIB),MAP=FULL:PART.          Prototype statement
REWIND,DN=$BLD:&SOURCE.
CAL,I=&SOURCE,L=&LIST,ABORT.
LDR,NX,LIB=&LIB,MAP=&MAP,L=&LIST,AB=$ABD.
REWIND,DN=$ABD:&LIST.                         Definition body
SAVE,DN=$ABD,PDN=MYPROGRAM.
IF(&GO.EQ.'YES')
$ABD.
ENDIF.
ENDPROC.
```

Invocation

```
BLDABS,WORK,,GO,LIB=VLIB2.
```

Expansion

```
REWIND,DN=$BLD:WORK.
CAL,I=WORK,L=,ABORT.
LDR,NX,LIB=VLIB2,MAP=FULL,L=.
REWIND,DN=$ABD:.
SAVE,DN=$ABD,PDN=MYPROGRAM.
IF('YES'.EQ.'YES')
$ABD.
ENDIF.
```

Example 3:

This procedure exemplifies the proper use of the underscore character for the definition of a formal parameter. It creates a procedure called AUDJCL.

Definition

```
PROC.
AUDJCL,DN,LEVEL,L=$OUT:AUDLST.              Prototype statement
AUDIT,PDN=&DN&LEVEL_JCL,ID=JCL,L=&L.        Definition body
ENDPROC.
```

Invocation

```
AUDJCL,-,05.
```

Expansion

```
AUDIT,PDN=-05JCL,ID=JCL,L=$OUT.
```

Example 4:

Parentheses are required when specifying multiple values for a single parameter value on a procedure definition prototype statement or on a calling statement. In these cases, the colon is used to separate default and Boolean values in a keyword parameter. For example:

Procedure-definition prototype statement

```
MYPROC,POS1,KEY=(DEF1:DEF2):(BOO1:BOO2).
```

Invocation

```
MYPROC,(POS1A:POS1B).
```

When substitution occurs during this call, POS1A:POS1B replaces all POS1 occurrences within the definition body. Both values (POS1A and POS1B) are evaluated separately during control statement evaluation. If apostrophes are on the call, 'POS1A:POS1B' is evaluated as one literal string.

Example 5:

The following procedure definition exemplifies the use of literal strings instead of parenthetical strings.

Definition

```
PROC.
PURGER,PDN,ID,ED,M.                                       Prototype
ACCESS,DN=$PURGE,PDN=&PDN,ID=&ID,ED=&ED,M=&M,UQ,NA.
DELETE,DN=$PURGE,NA.                                      Definition body
RELEASE,DN=$PURGE.
ENDPROC.
```

Invocation

```
PURGER,'SOURCE.MAIN',PROJECT.
```

Expansion

```
ACCESS,DN=$PURGE,PDN='SOURCE.MAIN',ID=PROJECT,ED=,M=,UQ,NA.
DELETE,DN=$PURGE,NA.
```

The apostrophes remain as part of the string in the expansion. If parentheses had been used in the invocation instead of apostrophes for the permanent dataset name, (SOURCE.MAIN), the value when the ACCESS statement is evaluated would be SOURCE.MAIN because the outermost parentheses are removed when preceded by a valid separator. This action would cause an error because the period in SOURCE.MAIN would be evaluated as a statement terminator during evaluation.

Example 6:

The following example illustrates the use of parenthetical strings instead of literal strings in a procedure definition.

Definition

```
PROC.
LGO,CALSORC,ABS,NLIB=$SCILIB:($SCILIB:
  $SYSLIB:$FTLIB).                              Prototype
CAL,I=&CALSORC.
LDR,NX,AB=&ABS,NOLIB=&NLIB.                     Definition body
ENDPROC.
```

Invocation

```
LGO,,,NLIB.
```

Expansion

```
CAL,I=.
LDR,NX,AB=,NOLIB=$SCILIB:$SYSLIB:$FTLIB.
```

Parentheses were not included for the expansion of the NLIB keyed default value because parentheses are removed during processing when preceded by the concatenation delimiter (:).

If apostrophes had been used instead of parentheses for the NLIB parameter value, the colons would have been ignored as separators during expansion. Also, apostrophes are treated as part of the value when included in a procedure definition prototype statement or a calling statement. Therefore, if apostrophes had been used, the following expansion would have occurred.

```
CAL,I=.
LDR,NX,AB=,NOLIB='$SCILIB:$SYSLIB:$FTLIB'.
```

When the LDR statement is executed, the value assigned to the NOLIB parameter is the literal string $SCILIB:$SYSLIB:$FTLIB which violates the syntax for the NOLIB parameter.

Example 7:

Consider the following procedure definition. This procedure is used to retrieve specified source decks from an UPDATE program library by the use of the &DATA option.

```
PROC.
GDECK,PLNAME,MASTERCH,DECKRNGE.          Prototype statement
ACCESS,DN=&PLNAME.
UPDATE,I=QZRRZQ2,Q,C=0,S,P=&PLNAME.
RELEASE,DN=QZRRZQ2:&PLNAME.              Definition body
&DATA QZRRZQ2
&MASTERCH_COMPILE &DECKRNGE
ENDPROC.
```

Two sample invocations and their expansions follow:

Invocation	Expansion
GDECK,COSPL,*,(ST,CT).	ACCESS,DN=COSPL. UPDATE,I=QZRRZQ2,Q,C=0,S,P=COSPL. RELEASE,DN=QZRRZQ2:COSPL. (Dataset QZRRZQ2 contains: *COMPILE ST,CT)
GDECK,FTLIBPL,*,(COS.RFD).	ACCESS,DN=FTLIBPL. UPDATE,I=QZRRZQ2,Q,C=0,S,P=FTLIBPL. RELEASE,DN=QZRRZQ2:FTLIBPL. (Dataset QZRRZQ2 contains: *COMPILE COS.RFD)

Example 8:

The example illustrates one mechanism for defining and maintaining user procedure libraries. Note the new procedure library is saved on mass storage for later use.

```
ACCESS,DN=GENLIB.
CALL,DN=GENLIB.
```

The permanent dataset GENLIB contains:

```
ECHO,OFF.
RELEASE,DN=$PROC.
*.
*.        Define procedure for ACCESS of commonly used ID.
*.
PROC.
UQ,DN,ED=:1,PDN=:GENLIB,R=:READCW,W=:WRITECW,M=:MAINCW,NA=:NA.
ACCESS,DN=&DN,ID=MYUID,PDN=&PDN,ED=&ED,R=&R,W=&W,M=&M,NA=&NA.
RETURN.
EXIT.
RETURN,ABORT.
ENDPROC.
*.
*.        Edit a local dataset.
*.
PROC.
ED,DN,AC=:'ACCESS'.
IF('&AC'.EQ.'ACCESS')
  UQ,&DN.
ENDIF
TEDI,DN=&DN.
RETURN.
EXIT.
RETURN,ABORT.
ENDPROC.
*.
*.        End of definitions
*.
UQ,PROCLIB,NA.
SAVE,DN=$PROC,PDN=PROCLIB,ID=MYUID.
DELETE,DN=PROCLIB,NA.
RELEASE,DN=$PROC.
ACCESS,DN=PROCLIB,ID=MYUID.
LIBRARY,DN=*:PROCLIB.
ECHO,ON.
```

APPENDIX SECTION

JOB USER AREA A

The table illustrations and their field descriptions are system generated.

JOB TABLE AREA - JTA

Each job has an area referred to as the Job Table Area (JTA) preceding the field defined for the user. A JTA is accessible to the operating system but not to the user. The format of a JTA is described in the COS Table Descriptions Internal Reference Manual, CRI publication SM-0045. The Job Table Area contains job-related information such as accounting data; a JXT pointer; sense switches; one or more task control blocks (TCBs), each of which contains an exchange package area and an area for saving B, T, and V register contents; control statement and logfile DSPs; and buffers; a copy of the user's LFTs; and a Dataset Name Table (DNT) for each dataset used by the job.

JOB COMMUNICATION BLOCK - JCB

Following the JTA is a 128-word block referred to as the Job Communication Block (JCB). The user accessible JCB contains a copy of the current control statement for the job and other job-related information.

Figure A-1 illustrates the JCB.

```
     0....+....1....+....2....+....3....+....4....+....5....+....6...

     +---------------------------------------------------------------+
   5 |                             CCI                               |
     |                                                               |
     |                                                               |
     |                                                               |
  17 |                                                               |
     +---------------------------------------------------------------+
  20 |                             CPR                               |
     |                                                               |
     |                                                               |
     |                                                               |
  77 |                                                               |
     +-------------------------------------------------------+-------+
 100 |                          JN                           |///////|
     +-------+--++-+-+-----------------------+---------------+-------+
```

Figure A-1. Job Communication Block (JCB)

```
    0....+....1....+....2....+....3....+....4....+....5....+....6...

    +-------+--++-+-+-----------------------+---------------+-------+
101 |  LPP  |//||/|U|          HLM          |          FL           |
    +-------+--++-+-+-----------------------+-----------------------+
102 |      NPF      |          BFB          |          DSP          |
    +---------------+-----------------------+-----------------------+
103 |      NLE      |          MFL          |          LFT          |
    ++++++++++++++++++----------------------+-------+-------+-------+
104 ||||||||||||||||||         ULFT         |       | PNST  | STRM  |
    ++++++++++++++++++---+--------+----+----+++-----+-+++++-+-------+
105 ||||||     CYCL      | CPTP   |MCP |NLCP|||  IAC  |||||/////////|
    ++++++---------------+--------+----+----+++-------+++++---------+
106 |                              CRL                              |
    +---------------------------------------------------------------+
107 |                              ACN                              |
    +                                                       +-------+
110 |                                                       |///////|
    +-------------------------------------------------------+-------+
111 |                              PWD                              |
    +                                                       +-------+
112 |                                                       |///////|
    +-------------------------------------------------------+-------+
113 |                              PROM                             |
    +---------------+---------------+---------------+---------------+
114 |     NULE      |     PLEV      |     ILEV      |     CLEV      |
    +---------------+---------------+---------------+---------------+
115 |             MMIN              |             MMIS              |
    +-------------------------------+-------------------------------+
116 |             MMBA              |             MMEP              |
    +------------------------------++-------------------------------+
117 |             STIN             ||             STIS              |
    +------------------------------++-------------------------------+
120 |///////////////////////////////|             AVBA              |
    +-------------------------------+-------------------------------+
121 |                              TSF                              |
    +-------------------------------+-------------------------------+
122 |              PSM              |///////////////////////////////|
    +-------------------------------+-------------------------------+
123 |                              DMM                              |
    +---------------------------------------------------------------+
124 |///////////////////////////////////////////////////////////////|
    +                                                               +
125 |///////////////////////////////////////////////////////////////|
    +---------------+-----------------------------------------------+
126 |     NUDP      |///////////////////////////////////////////////|
    +---------------+               +-------------------------------+
127 |///////////////////////////////|             TPTR              |
    +-------------------------------+-------------------------------+
```

Figure A-1. Job Communication Block (JCB) (continued)

```
     0....+....1....+....2....+....3....+....4....+....5....+....6...

     +-------------------------+-------------------------+
130  |                        RDYQ                       |
     +---------------------------------------------------+
131  |                        RUNQ                       |
     +---------------------------------------------------+
132  |///////////////////////////////////////////////////|
     |                                                   |
     |///////////////////////////////////////////////////|
     |                                                   |
145  |///////////////////////////////////////////////////|
     +---------------------------------------------------+
146  |                        LDR                        |
     |                                                   |
     |                                                   |
     |                                                   |
163  |                                                   |
     +---------------------------------------------------+
164  |                        STN                        |
     +---------------------------------------------------+
165  |                        STF                        |
     +---------------------------------------------------+
166  |                        BDAT                       |
     +---------------------------------------------------+
167  |                        BTIM                       |
     +---------------------------------------------------+
170  |                        DIG                        |
     |                                                   |
     |                                                   |
     |                                                   |
177  |                                                   |
     +---------------------------------------------------+
200  |///////////////////////////////////////////////////|
     +---------------------------------------------------+
```

Figure A-1. Job Communication Block (JCB) (continued)

Field	Word(base8)	Bits	Description
	0-5		The first five words of the JCB are assigned as a save area for the BGN table that is used by F$BGN.
JCCCI	5-17	0-63	Control statement image packed 8 characters per word
JCCPR	20-77	0-63	Control statement parameters, expanded to two words per parameter

Field	Word(base8)	Bits	Description
JCJN	100	0-55	Job name; bits 56-63 must be 0.
JCLPP	101	0-7	Lines per page
JCRMSG	101	11	RFL message sent
JCU	101	14-15	User mode indicator:
JCUL	101	14	Local
JCUG	101	15	Global
JCHLM	101	16-39	High limit of user code
JCFL	101	40-63	Current field length
JCNPF	102	0-15	Number of physical buffers and datasets
JCBFB	102	16-39	Base address of I/O buffers
JCDSP	102	40-63	Base address of DSP area
JCNLE	103	0-15	Number of entries in LFT
JCMFL	103	16-39	Maximum FL allowed
JCLFT	103	40-63	Base of LFT
JCDCS	104	0	CSP dynamic control statement flag
JCCSDB	104	1	CSP debug flag
JCBP	104	2	JOB statement breakpoint (BP) flag
JCNTB	104	3	CSP traceback suppression flag
JCIOAC	104	4	I/O area current status flag: 0 User's I/O area is unlocked 1 User's I/O area is locked
JCIOAP	104	5	I/O area previous status flag: 0 User's I/O area is unlocked 1 User's I/O area is locked
JCIA	104	6	Interactive flag
JCCHG	104	7	Execute CHARGES utility for trailer message.

Field	Word(base8)	Bits	Description
JCJBS	104	8	JOB statement flag (if set, JOB statement just processed)
JCCSIM	104	9	Flag is set when CRAY-1 simulator is running.
JCDLIT	104	10	Display literal delimiters in control statement crack.
JCRPRN	104	11	Retain level 1 parentheses.
JCVSEP	104	12	Last character was valid separator.
JCSDM	104	13	NOECHO of current control statement
JCPDMS	104	14	Suppress PDM user logfile messages
JCCSQ	104	15	New CFT calling sequence in effect
JCOVT	104	16	Overlay type
JCULFT	104	17-47	Base of user LFTs (JCB-REL)
JCPNST	104	48-55	Parentheses nesting level for current control statement
JCSTRM	104	56-63	Statement termination for current control statement
JCEFI	105	0	Enable floating-point interrupt flag; used by $ARLIB math routines to reset floating-point interrupt flag
JCOVL	105	1	Overlay flag
JCSBC	105	2	SBCA flag
JCBDM	105	3	Enable bidirectional mode flag
JCORI	105	4	Interrupt on operand range flag
JCCYCL	105	5-20	CPU cycle time, in picoseconds
JCCPTP	105	21-29	CPU type, @CRAYxxx
JCMCP	105	30-34	Maximum number of logical CPUs that can be assigned to a user job

Field	Word(base8)	Bits	Description
JCNLCP	105	35-39	Current number of logical CPUs asg'd
JCEMA	105	40	1=Extended memory addressing enabled
JCAVL	105	41	1=Additional vector logical unit enab.
JCIAC	105	42-49	Number of account processing retries allowed for an interactive job
JCACRQ	105	50	Accounting mandatory flag
JCPWRQ	105	51	Password mandatory flag
JCRYPT	105	52	Encryption flag
JCSLVL	105	53	Security level flag
JCCRL	106	0-63	COS revision level
JCCRLS	106	32-63	COS revision number
JCACN	107-110	0-63	1 through 15 character account number
JCACN1	107	0-63	Characters 1 through 8 of account number
JCACN2	110	0-55	Characters 9 through 15 of account number
JCPWD	111-112	0-63	1 through 15 character password
JCPWD1	111	0-63	Characters 1 through 8 of password
JCPWD2	112	0-55	Characters 9 through 15 of password
JCPROM	113	0-63	Current user job interactive prompt, justified, zero-filled. 64 bits of binary zeroes disables user job prompt. Set to system default at beginning of each job step.
JCNULE	114	0-15	Number of user LFT entries (below HLM)
JCPLEV	114	16-31	Current procedure nesting level
JCILEV	114	32-47	Current iterative nesting level
JCCLEV	114	48-63	Current conditional nesting level

Field	Word(base8)	Bits	Description
			The next four words are used by the run-time memory manager:
JCMMIN	115	0-31	Size of increments to the managed memo
JCMMIS	115	32-63	Initial size of memory to be managed
JCMMBA	116	0-31	Base address of managed space
JCMMEP	116	32-63	Size of smallest block added to available space list
JCSTIN	117	0-30	Size of increments to a stack
JCSTRT	117	31	Flag to indicate stack for root task
JCSTIS	117	32-63	Initial size of a stack
JCAVBA	120	32-63	Base of available space
JCTSF	121	0-63	Task scheduling flag
JCPSM	122	0-31	Pseudo semaphore registers 1 A&B, 1/S
JCDMM	123	0-63	Don't move memory when nonzero
JCNUDP	126	0-15	Number of system DSPs in user
JCTPTR	127	32-63	Pointer to list of all tasks
JCRDYQ	130	0-63	Multitasking ready queue header
JCRUNQ	131	0-63	Multitasking run queue header
JCLDR	146-163	0-63	Unsatisfied externals
JCSTN	164	0-63	Job step count
JCSTF	165	0-63	Job step failure flag
JCBDAT	166	0-63	Date of absolute load module generation
JCBTIM	167	0-63	Time of absolute load module generation
JCDIG	170-177	0-63	Reserved for diagnostics

LOGICAL FILE TABLE - LFT

The Logical File Table contains a 2-word entry for each dataset name and each alias for a dataset. Each entry points to the DSP for a dataset. Figure A-2 illustrates an LFT for a dataset.

```
     0....+....1....+....2....+....3....+....4....+....5....+....6...

     +----------------------------------------------------------+--------+
   0 |                             DN                           |////////|
     +---+--------------------------------------+---------------+--------+
   1 |OST|//////////////////////////////////////|          DSP           |
     +---+--------------------------------------+------------------------+
```

Figure A-2. Logical File Table (LFT) entry

Field	Word(base8)	Bits	Description
LFDN	0	0-55	Dataset name
LFOST	1	0-3	DATASET OPEN STATUS
LFDSP	1	40-63	DSP address

DATASET PARAMETER AREA - DSP

Information concerning the status of a particular dataset and location of the I/O buffer for the dataset is maintained in the Dataset Parameter Area (DSP) of the user field. The DSP is illustrated in figure A-3.

```
       0....+....1....+....2....+....3....+....4....+....5....+....6...

       +-----------------------------------------------------+--------+
    0  |                          DN                         |////////|
       ++-----------+--+---------+++---++-+++++++----------------+--------+
    1  ||    ERR    ||*|   BFI   |||OST||*|||||||          FRST           |
       ++---------+--++-+---------+++---++-+++++++----------------------+
    2  |/////////  IBP          IBN                        IN             |
       ++-+------+-----+-----------------------+----------------------+
    3  ||/| RBC  | OBP |          OBN          |          OUT         |
       ++-+------+-----+-----------------------+----------------------+
    4  || |     BS     |          TBN          |          LMT         |
       ++++++----------+--------+--------------+----------------------+
    5  ||||||        PFI        |      PRI     |          RCW         |
       ++++++-------------------+--------------+----------------------+
    6  |||                           LPW                              |
       +++------+-----+-----------------------+----------------------+
    7  ||| BF   |BUBC |          BWC          |          BWA         |
       +++------+-----+-----------------------+----------------------+
   10  |                             TM                               |
       |                                                              |
       |                                                              |
       |                                                              |
   17  |                                                              |
       +--------------+-----------------------+----------------------+
   20  |     TPS      |///////////////////////|          TPV         |
       +-+-+----------+---+-------------------+-------+--------------+
   21  |*|*|    TAPE  |   |///////////////////////////|     MTF      |
       +++-++------+---+---+-------------------+-------+--------------+
   22  ||FD |  RF  |///|          MBS          |          RS          |
       ++---++-+---+-+++-----------------------+----------------------+
   23  |BFBO |*|///|*|||         BFBL          |         BFBA         |
       +-----+++---+-+++-----------------------+----------------------+
   24  |LPBL |||///////|          SBL          |         BLBL         |
       +-----+++-------+-----------------------+----------------------+
   25  |                            LOCK                              |
       +-----------+--------------------------------------------------+
   26  |    EEC    |//////////////////////////////////////////////////|
       +-----------+++++-----------------------+----------------------+
   27  |///////////|||||         RECL          |         NXRC         |
       +-----------+++++-----------------------+----------------------+
```

Figure A-3. Dataset Parameter Area

Field	Word(base8)	Bits	Description
DPDN	0	0-55	Dataset name
DPBSY	1	0	Busy flag, circular I/O: 0 Not busy 1 Busy
DPERR	1	1-12	Error flags:
DPEOI	1	1	End of data on read; write past allocated disk space on write.
DPENX	1	2	Dataset does not exist
DPEOP	1	3	Dataset not open
DPEPD	1	4	Invalid processing direction
DPEBN	1	5	Block number error
DPEDE	1	6	Unrecovered data error
DPEHE	1	7	Unrecovered hardware error
DPERW	1	8	Attempted read after write or past EOD
DPEPT	1	9	Dataset prematurely terminated
DPELE	1	10	Unrecovered logical data error Reserved
DPEEP	1	12	Extended error (see DPEEC)
DPSTS	1	14-15	Status: 00 Closed 01 Open for output (O) 10 Open for input (I) 11 Open for I/O
DPBFI	1	16-24	Blank compression character in ASCII (BFI=O'777 implies no compression)
DPQIO	1	26	Queued I/O Request Flag
DPOST	1	27-30	Open status
DPABD	1	31	Accept bad data flag
DPTP	1	32-33	Tape dataset (online/staged)
DPTRAN	1	34	Transparent mode for interactive dataset
DPIA	1	35	Dataset is interactive
DPMEM	1	36	Dataset is memory resident

Field	Word(base8)	Bits	Description
DPRDM	1	37	Random dataset flag: 0 Sequential dataset 1 Random dataset
DPUDS	1	38	Undefined dataset structure: 0 COS-blocked dataset structure 1 Undefined dataset structure
DPEND	1	39	Write end-of-data flag
DPFRST	1	40-63	Address of first word of buffer
DPIBP	2	10-15	Input bit position
DPIBN	2	16-39	Block number, read request. System reads from block number until buffer is filled. DPIBN is then set to the next block number.
DPIN	2	40-63	Address of current input word
DPSPOS	3	0	Asynchronous SETPOS busy flag
DPRBC	3	3-9	Remaining blank count
DPOBP	3	10-15	Bit position in current output word (character I/O only)
DPOBN	3	16-39	Block number, write request. System writes from block number until buffer is empty. The next block number is then in DPOBN.
DPOUT	3	40-63	Address of current output word
DPUEOF	4	0	Uncleared end-of-file (EOF)
DPBS	4	1-15	Buffer size (in D'512 word sectors)
DPTBN	4	16-39	Temporary block number; used by random I/O for last block read
DPLMT	4	40-63	Address of last word+1 of buffer. LMT minus FRST defines buffer size.
DPEOR	5	0	EOR flag
DPEOF	5	2	EOF flag

Field	Word(base8)	Bits	Description
DPEOD	5	3	EOD flag
DPRW	5	4	Previous operation read/write flag: 0 Read 1 Write
DPPFI	5	5-24	Previous file index; backward index to block containing previous EOF.
DPPRI	5	25-39	Previous record index; backward index to block containing previous EOR.
DPRCW	5	40-63	Control word address: Previous RCW address if in write mode Next RCW if in read mode
DPLPW	6	0-63	Last partial word; used for character mode I/O
DPBIO	7	0	Buffered I/O busy: 0 Buffered I/O operation complete 1 Buffered I/O operation incomplete
DPBER	7	1	Buffered I/O error flag
DPBF	7	2-9	Function code: BIOFRRP = 0 Read partial record BIOFRR = O'10 Read record BIOFWRP = O'40 Write partial record BIOFWR = O'50 Write record BIOFEOF = O'52 Write EOF BIOFEOD = O'56 Write EOF
DPBPD	7	4	Processing direction: 0 Read 1 Write
DPBEO	7	6-9	Termination condition: 00 Partial 10 Record 12 File, write only 16 Dataset, write only
DPBUBC	7	10-15	Unused bit count; must be specified on a write record request. Value returned on a read request.

Field	Word(base8)	Bits	Description
DPBWC	7	16-39	Word count; number of words at DPBWA to read or write. Field contains actual number of words read when request is completed.
DPBWA	7	40-63	Word address of user data area L@DPTM=D'8
DPTM	10-17	0-63	

Wd	Bits	Use
8	0-63	Saved word W@DPPRI
9	0-63	Saved A2 in WB30
10	16-39	$RWDP/$WWDP return address
10	40-63	$RWDP/$WEOF first word address (FWA)
11	16-39	WB30/$WEOF return address
11	40-63	$WEOD return address
12	0-7	JTA length/1000 octal when registers are saved
12	8-15	Bits 0-7 of RBLK/WBLK A5
12	16-39	(B.ZE)
12	40-63	RBLK/WBLK B0
13	16-39	DNT address
13	40-63	(A7) JXT address recall
14	0-15	Bits 8-23 of RBLK/WBLK A5
14	16-39	RBLK/WBLK A2
14	40-63	RBLK/WBLK A3
15	0-63	RBLK/WBLK S6

Field	Word(base8)	Bits	Description
DPTPS	20	0-15	Online tape status
DPTPV	20	40-63	Tape pointer to label definition table
DPTPD	21	0-1	Tape density
DPTPF	21	2-3	Tape format

Field	Word(base8)	Bits	Description
DPTAPE	21	4-19	Tape status
DPAEV	21	4	User is at tape end of volume
DPTOR	21	5	Tape off reel
DPTMS	21	6	Tape mark status
DPBLT	21	7	Blank tape
DPEOVR	21	8	EOV READ

MASKS FOR TESTING TAPE STATUS FIELD

TS$EOV=O'100000	EOV mask
TS$TOR=O'040000	Tape off reel mask
TS$TMS=O'020000	Tape mark status mask
TS$BLT=O'010000	Blank tape detected mask
TS$EOVR=O'004000	Read completed in EOV processing

DPMTF	21	48-63	Maintenance test field
DPCV	22	0	Data conversion flag DPCVOFF=0 Data conversion off DPCVON=1 Data conversion on
DPFD	22	1-4	Translation identifier DPFDNONE=0 NO foreign file translation DPFDIBM=1 IBM file translation DPFDCDC=2 CDC file translation
DPRF	22	5-11	Record format (if DPCT nonzero)

DPRFUNKN=O'177	Unknown record format
DPRFIU=0	IBM undefined
DPRFIF=1	IBM fixed
DPRFIFB=2	IBM fixed blocked
DPRFIV=3	IBM variable
DPRFIVB=4	IBM variable blocked
DPRFIVBS=5	IBM variable block span

Field	Word(base8)	Bits	Description
			Values 21 through 37 are reserved for ANSI record types:
			DPRFIIW=O'00 I tape format, I blocks, W records
			DPRFICW=O'10 I tape format, C blocks, W records
			DPRFICZ=O'11 I tape format, C blocks, Z records
			DPRFICS=O'12 I tape format, C blocks, S records
			DPRFSIIW=O'40 SI tape format, I blocks, W records
			DPRFSICW=O'50 SI tape format, C blocks, W records
			DPRFSICZ=O'51 SI tape format, C blocks, Z records
			DPRFSICS=O'52 SI tape format, C blocks, S records
DPMBS	22	16-39	Maximum block size
DPRS	22	40-63	Record length
DPBFBO	23	0-5	User data area current bit offset
DPCS	23	6-7	Character set (if DPCT nonzero) DPCSAS=0 ASCII DPCSEB=1 EBCDIC DPCSDC=2 CONTROL DATA display code
DPSCC	23	12-13	Record continuation code
DPBDF	23	14	Bad data flag
DPPCR	23	15	Process-characters-remaining flag
DPBFBL	23	16-39	User data area current bit length
DPBFBA	23	40-63	User data area current address
DPLPBL	24	0-5	Last partial word bit length
DPEOLR	24	6	Foreign dataset end of logical record
DPEOLF	24	7	Foreign dataset end of logical file
DPSBL	24	16-39	Current segment/record bit length
DPBLBL	24	40-63	Current tape block bit length

Field	Word(base8)	Bits	Description
DPLOCK	25	0-63	Multitasking lock (nonzero TIB address if log is set)
DPEEC	26	0-11	Error code if DPEEP is set; correspond to EXP abort codes.
DPDEL	27	12	FORTRAN file status: 0 Keep 1 Delete
DPBLNK	27	13	FORTRAN numeric input blank conversion: 0 Null 1 Zero
DPDIR	27	14	FORTRAN direct access flag
DPUFMT	27	15	FORTRAN unformatted I/O flag
DPRECL	27	16-39	FORTRAN direct access record length (in number of characters)
DPNXRC	27	40-63	FORTRAN direct access next record number

When the FD field in word 22 of the Dataset Parameter Area is equal to CDC, it is redefined as shown in figure A-4.

```
    0....+....1....+....2....+....3....+....4....+....5....+....6...

    +----+-+-+--+---------------------------------------------------+
 22 |////|F|*|RT|///////////////////////////////////////////////////|
    +----+-+-+--+---------------------------------------------------+
```

Figure A-4. CDC record format

Field	Word(base8)	Bits	Description
DPF	22	5-6	Tape format DPFI=0 Internal DPFSI=1 System or scope internal
DPBT	22	7-8	Block type DPBTI=0 Internal DPBTC=1 Character count

Field	Word(base8)	Bits	Description	
DPRT	22	9-11	Record type	
			DPRTW=0	Control word
			DPRTZ=1	Zero byte
			DPRTS=2	System-logical

Figure A-5 shows the redefinition of the LPW field in word 6 of the Dataset Parameter Area used by asynchronous SETPOS as save areas.

```
     0....+....1....+....2....+....3....+....4....+....5....+....6...

     +---------------------------------------------------------------+
   0 |///////////////////////////////////////////////////////////////|
     |                                                               |
     |///////////////////////////////////////////////////////////////|
     |                                                               |
   5 |///////////////////////////////////////////////////////////////|
     +                     +--------+--------------------------------+
   6 |/////////////////////| SPWA   |              SPIN              |
     +---------------------+--------+--------------------------------+
```

Figure A-5. Save areas used by asynchronous SETPOS

Field	Word(base8)	Bits	Description
DPSPWA	6	22-30	Word address save areas used
DPSPIN	6	31-63	by asynchronous SETPOS

PERMANENT DATASET DEFINITION TABLE - PDD

The PDD is a parameter list that gives input to the Permanent Dataset Manager. The PDD illustrated in figure A-6 is used for all save, access, dump access, load, modify, permit, rewrite SDT, pseudo-access, and permanent dataset name requests. The PDD illustrated in figure A-7 is used for both DSC and DXT page requests, and for dump time requests. The PDD illustrated in figure A-8 is used for all delete, release, and adjust requests. The PDD illustrated in figure A-9 is used for queue and dequeue SDT requests, and for get and link DXT requests. Table A-1 presents a list of Permanent Dataset function codes.

Table A-1. Permanent dataset function codes

Symbol	Octal Code	Function
PMFCSU	10	Save user dataset
PMFCSI	12	Save input dataset
PMFCSO	14	Save output dataset
PMFCAU	20	Access user dataset
PMFCAI	26	Access spooled dataset
PMFCAO	26	Access spooled dataset
PMFCDU	30	Delete user dataset
PMFCDI	36	Delete spooled dataset
PMFCDO	36	Delete spooled dataset
PMFCPG	40	DSC Page request
PMFCPX	41	DXT Page request
PMFCLU	50	Load user dataset
PMFCLI	52	Load input dataset
PMFCLO	54	Load output dataset
PMFCRL	60	PDS/Release request
PMFCPN	70	PDN request
PMFCDT	100	Dump time request
PMFCDQ	110	Dequeue SDT
PMFCEA	120	Queue SDT to available queue
PMFCEI	122	Queue SDT to input queue
PMFCEO	124	Queue SDT to output queue
PMFCAD	130	Adjust user dataset
PMFCMD	140	Modify user dataset
PMFCRSDT	150	Rewrite input SDT entry
PMFCPSAC	160	Pseudo-access for RRJ
PMFCPU	170	Access user saved dataset for PDSDUMP
PMFCPO	176	Access output dataset for PDSDUMP
PMFCPI	176	Access input dataset for PDSDUMP
PMFCPE	200	Permit Request
PMFCLKDX	210	Link DXT Request
PMFCRTDX	220	Get DXT Request

```
   0....+....1....+....2....+....3....+....4....+....5....+....6...

   ++++++-+-+-++++------------------+-------+-----------+-----------+
 0 ||||||*|*|*||||//////////////////| SIZE  |    ST     |    FC     |
   ++++++-+-+-++++------------------+-------+-----------+---+-------+
 1 |                      DN                                |///////|
   +--------------------------------------------------------+-------+
 2 |                          PDN                                   |
   +                                                        +-------+
 3 |                                                        |///////|
   +--------------------------------------------------------+-------+
 4 |                          ID                                    |
   +----------------------------------------------------------------+
 5 |                          USR                                   |
   +                                                        +-------+
 6 |                                                        |///////|
   +-----------------------+---------------+-----------+---+-------+
 7 |          TXT          |       FM      |     RT    |    ED     |
   +-----------------------+---------------+-----------+---+-------+
10 |                      OJB                               |///////|
   +---------------+---------------+---------------+---+-------+
11 |      SID      |      DID      |      DC       |      JSQ      |
   +---------------+---------------+---------------+---------------+
12 |                          TID                                   |
   +----------------------------------------------------------------+
13 |                          SF                                    |
   ++++-------++++++---------------+-----------------------+-------+
14 |||| TXL  ||||||      MFL      |           TL          |  PR   |
   ++++-------++++++---------------+-----------------------+-------+
15 |                          RD                                    |
   +----------------------------------------------------------------+
16 |                          WT                                    |
   +----------------------------------------------------------------+
17 |                          MN                                    |
   +--------------------------------------------------------+-------+
20 |                      JCN                               |///////|
   +--------------------------------------------------------+-------+
21 |                      CL                                |///////|
   ++-------+---------------+-----------------------+-----++-------+
22 || JSP   |      JCR      |          OLM          |RJST || IJSP  |
   +++--+-+-+------++--------+--------------+--------+-----++-------+
23 |*|**|*|///////||          TPB           |         TPV          |
   +++-++++-+------++-----------------------+----------------------+
24 ||**|||**|////////////////////////////////////////////////////////|
   ++--+++--+-------------------------------------------------------+
25 ||//////////////////////////////////////////////////////////////|
   ++--------------------------------------------------------------+
26 ||//////////////////////////////////////////////////////////////|
   +++----------------------------------------------+---------------+
```

Figure A-6. Permanent Dataset Definition Table (PDD)

```
   0....+....1....+....2....+....3....+....4....+....5....+....6...

   +++---------------------------------------------+---------------+
27 |*|/////////////////////////////////////////////|      TXO      |
   +++-----+-----------------------+---+-----------+---------------+
30 ||//////|          LSD          |///|            FPE            |
   ++------+-------+---------------+---+-----------+---------------+
31 |      ACS      |              DSZ              |     OJSQ      |
   +---------------+-------------------------------+---------------+
32 |                              CRT                              |
   +---------------------------------------------------------------+
33 |                              ACT                              |
   +---------------------------------------------------------------+
34 |                              TDM                              |
   +---------------------------------------------------------------+
35 |                              MOD                              |
   +-------+-------+-----------+-----------------------------------+
36 |  SSC  |  TXC  |    MML    |///////////////////////////////////|
   +++-+---+-------+-----------+---+-------------------------------+
37 |||*|///|  PAM  |     ADNM      |///////////////////////////////|
   +++-+---+-------+---------------+-----------------------+-------+
40 |                         ADN                           |///////|
   +-------+-----------------------+-----------------------+-------+
41 | NOTL  |         NOTE          |///////////////////////////////|
   +-------+-----------------------+-------------------------------+
42 |                              CHG                              |
   +---------------------------------------------------------------+
43 |                              OWN                              |
   +                                                       +-------+
44 |                                                       |///////|
   +-------------------------------------------------------+-------+
45 |                              DNS                              |
   +---------------------------------------------------------------+
46 |                              ACN                              |
   +                                                       +-------+
47 |                                                       |///////|
   +-------------------------------------------------------+-------+
```

Figure A-6. Permanent Dataset Definition Table (PDD) (continued)

Field	Word(base8)	Bits	Description
PMSG	0	0	Normal completion message suppression indicator
PMERR	0	1	Error message suppression indicator
PMWAIT	0	2	WAIT flag for a disposed dataset

Field	Word(base8)	Bits	Description
PMNRLS	0	3	No release of dataset on DISPOSE
PMAQR	0	4	Acquire flag for accounting
PMTP	0	5-6	Tape dataset (online/staged)
PMTCS	0	7-8	Tape dataset character set
PMEXO	0	9-10	Execute only
PMDTR	0	11	Update dump-time on PDSDUMP access
PMSMT	0	12	Submit flag
PMDFFL	0	13	Job-used-MFL-default flag
PMSIZE	0	32-39	PDD size in words
PMST	0	40-51	Return status
PMFC	0	52-63	Function code (see chart PM-1)
PMDN	1	0-55	Local dataset name
PMPDN	2-3	0-63	Permanent dataset name
PMPDN1	2	0-63	Characters 1-8
PMPDN2	3	0-55	Characters 9-15
PMID	4	0-63	User identification
PMUSR	5-6	0-63	User number
PMUSR1	5	0-63	Characters 1-8
PMUSR2	6	0-55	Characters 9-15
PMTXT	7	0-23	Address of optional text field
PMFM	7	24-39	Format designator (two characters): FMCD=CD Character/deblocked FMCB=CB Character/blocked FMBD=BD Binary/deblocked FMBB=BB Binary/blocked
PMRT	7	40-51	Retention period; 0-4095 days.
PMED	7	52-63	Edition number (0-4095)

Field	Word(base8)	Bits	Description
PMOJB	10	0-55	Originating job name
PMSID	11	0-15	Source ID; 2 characters.
PMDID	11	16-31	Destination ID; 2 characters.
PMDC	11	32-47	Disposition code; 2 characters. DCIN=IN Job dataset DCST=ST Dataset to be staged DCSC=SC Scratch dataset DCPR=PR Print dataset DCPU=PU Punch dataset DCPT=PT Plot dataset DCMT=MT Magnetic tape dataset
PMJSQ	11	48-63	Job sequence number
PMTID	12	0-63	Terminal ID; 1-8 characters.
PMSF	13	0-63	Special forms
PMUQ	14	0	Unique ACCESS/Write ring requested
PMENT	14	1	Enter in System Directory
PMIR	14	2	Immediate reply requested
PMTXL	14	3-10	Number of words of text
PMNRR	14	11	Job rerun flag; set if job cannot be rerun (input entries only).
PMINIT	14	12	Job initiate flag; set if job has been initiated.
PMIA	14	13	Interactive flag
PMDFR	14	14	Deferred disposition indicator
PMNA	14	15	No abort flag. If set, processing continues even if an error is encountered.
PMMFL	14	16-31	MFL parameter from job card (input)
PMSGFL	14	16	All available memory requested
PMFL	14	17-31	Field length/512
PMTL	14	32-55	Time limit (input datasets)

Field	Word(base8)	Bits	Description
PMPR	14	56-63	Priority (input datasets)
PMRD	15	0-63	Read permission control word
PMWT	16	0-63	Write permission control word
PMMN	17	0-63	Maintenance permission control word
PMJCN	20	0-55	Job class name
PMCL	21	0-55	CL parameter from JOB statement
PMSYS	22	0	System job
PMJSP	22	1-8	JOB statement priority
PMJCR	22	9-24	Job class rank
PMOLM	22	25-48	Size of $OUT in 512-word block
PMRJST	22	49-54	Job status flag
PMIJSP	22	56-63	Original job card priority
PMTPD	23	0-1	Tape density
PMTPL	23	2-4	Tape label type
PMTPF	23	5-6	Tape format
PMTPC	23	15	Tape cataloged dataset
PMTPB	23	16-39	Tape maximum block size in bytes
PMTPV	23	40-63	Tape pointer to label definition table
PMTPM	24	0	Tape online maintenance access
PMTPP	24	1-3	Tape parallel device count
PMTP2	24	4	Tape second device assignment
PMTPH	24	5	Tape hold assigned device
PMIDC	24	6-8	Tape initial disposition code
PM2164	25	0	Unused
PM2264	26	0	Unused

Field	Word(base8)	Bits	Description
PMTSCV	27	0-1	Timestamp conversion specification TSCVTHIS=0 Convert to current COS system TSCVRT=1 Convert to RT-based timestamp TSCVNS=2 Convert to NS-based timestamp TSCVSAME=3 No conversion -- leave timestamp alone
PMTXO	27	48-63	TXT ORDINAL OF USER TASK
PMOCC	30	0	Operator-changed-class flag
PMLSD	30	8-31	Temporary SDT address for load input/output
PMFPE	30	36-63	First DSC page/entry for dataset
PMFPP	30	36-59	First DSC page for dataset
PMFEN	30	60-63	First entry for dataset
PMACS	31	0-15	Number of accesses (load saved datasets only)
PMDSZ	31	16-47	Size of dataset as reflected by DSC DAT bodies (used only when a pseudo access is performed during the recovery of rolled jobs)
PMOJSQ	31	48-63	Originating job sequence number
PMCRT	32	0-63	Creation time in cycles (load request only)
PMACT	33	0-63	Time of last access in cycles (load request only)
PMTDM	34	0-63	Time of last dump in cycles (load request only)
PMMOD	35	0-63	Time of last modification in cycles (load request only)
PMSSC	36	0-7	Station slot word length

Field	Word(base8)	Bits	Description
PMTXC	36	8-15	Text field word length
PMMML	36	16-27	Interactive maximum message length
PMPDE	37	0	Partial delete flag
PMREM	37	1	Remove permit flag
PMTRA	37	2-3	Track accesses flag: TRAKNO=1 Do not track accesses TRAKYE=2 Do track accesses
PMPAM	37	8-15	Public/permit access mode: PAMEX=O'011 Execute only PAMRE=O'001 Read permission PAMWR=O'002 Write permission PAMMA=O'004 Maintanence permission PAMNO=O'200 No permissions MAXPAM=5
PMADNM	37	16-31	ADN propagate attributes mask: PACW=O'000001 Control words PAPAM=O'000002 Public access mode PATRK=O'000004 Track accesses PAPER=O'000010 Permits PATXT=O'000020 Text PANTS=O'000040 Notes PAALL=O'000077 All of the above PANO=O'100000 None MAXPA=D'8 Maximum allowable attributes
PMADN	40	0-55	Attributes dataset name
PMNOTL	41	0-7	Notes length in words
PMNOTE	41	8-31	Pointer to notes text LE@NOTE=D'60 Allow 480 characters for notes
PMCHG	42	0-63	Last modification time (PDSLOAD)
PMOWN	43-44	0-63	Dataset Owner

Field	Word(base8)	Bits	Description
PMOWN1	43	0-63	Owner (char 1-8)
PMOWN2	44	0-55	Owner (char 9-15)
PMDNS	45	0-63	Reserved for installation
PMACN	46-47	0-63	Account Number
PMACN1	46	0-63	Characters 1-8 of account number
PMACN2	47	0-55	Characters 9-15 of account number

```
  0....+....1....+....2....+....3....+....4....+....5....+....6...

  +++-----------------------------+-------+-----------+-----------+
0 |||/////////////////////////////| SIZE  |    ST     |    FC     |
  +++-----------------------------+-------+-----------+---+-------+
1 |                            DN                         |///////|
  +-------------------------------+---------------+-------+-------+
2 |///////////////////////////////|      NPG      |      BPG      |
  +-------+---------------+-------+-------+-------+---------------+
3 |///////|      NHP      |      NOP      |          BUF          |
  +-------+---------------+---------------+-----------------------+
```

Figure A-7. Permanent Dataset Definition Table (PDD) format 2

Field	Word(base8)	Bits	Description
PMSG	0	0	Normal completion message suppression indicator
PMERR	0	1	Error message suppression indicator
PMSIZE	0	32-39	PDD size in words
PMST	0	40-51	Return status
PMFC	0	52-63	Function code (see chart PM-1)
PMDN	1	0-55	Local Dataset Name (PMFCDT)
PMNPG	2	32-47	Number of pages (PMFCPG,PMFCPX)
PMBPG	2	48-63	Beginning page number (PMFCPG,PMFCPX)
PMNHP	3	8-23	Number of hash pages (returned by PDM for PMFCPG requests)

Field	Word(base8)	Bits	Description
PMNOP	3	24-39	Number of overflow pages (returned by PDM for PMFCPG requests)
PMBUF	3	40-63	Buffer address

```
   0....+....1....+....2....+....3....+....4....+....5....+....6...

   +++-----------------------------+-------+-----------+-----------+
 0 |||/////////////////////////////| SIZE  |    ST     |    FC     |
   +++-----------------------------+-------+-----------+---+-------+
 1 |                    DN                                 |///////|
   +-------------------------------------------------------+-------+
```

Figure A-8. Permanent Dataset Definition Table (PDD) format 3

Field	Word(base8)	Bits	Description
PMSG	0	0	Normal completion message suppression indicator
PMERR	0	1	Error message suppression indicator
PMSIZE	0	32-39	PDD size in words
PMST	0	40-51	Return status
PMFC	0	52-63	Function code (see chart PM-1)
PMDN	1	0-55	Local dataset name

```
   0....+....1....+....2....+....3....+....4....+....5....+....6...

   +++-----------------------------+-------+-----------+-----------+
 0 |||/////////////////////////////| SIZE  |    ST     |    FC     |
   +++----------+------------------+-------+-----------+-----------+
 1 |///////////|         DSPE              |          DXT          |
   +-----------+---------------------------+-----------------------+
 2 |                            PDN                                |
   +                                                       +-------+
 3 |                                                       |///////|
   +-----------------------+-----------------------+-------+-------+
 4 |///////////////////////|          SDT          |      SQJ      |
   +-----------------------+-----------------------+---------------+
```

Figure A-9. Permanent Dataset Definition Table (PDD) format 4

Field	Word(base8)	Bits	Description
PMSG	0	0	Normal completion message suppression indicator
PMERR	0	1	Error message suppression indicator
PMSIZE	0	32-39	PDD size in words
PMST	0	40-51	Return status
PMFC	0	52-63	Function code (see chart PM-1)
PMDSPE	1	12-39	Page/entry of main DSC entry (PMFCLKDX, PMFCRTDX requests)
PMDSP	1	12-35	Page number of main DSC entry (PMFCLKDX, PMFCRTDX requests)
PMDSE	1	36-39	Entry number of main DSC entry (PMFCLKDX, PMFCRTDX requests)
PMDXT	1	40-63	Pointer to DXT information buffer (PMFCLKDX, PMFCRTDX requests)
PMPDN	2-3	0-63	Permanent dataset name
PMPDN1	2	0-63	Characters 1-8
PMPDN2	3	0-55	Characters 9-15
PMSDT	4	24-47	SDT address Returned by PDM for PMFCDQ request Input for PMFCEA, PMFCEI, PMFCEO
PMSQJ	4	48-63	Job sequence number (PMFCDQ request)

BEGIN CODE EXECUTION TABLE - BGN

The BGN Table, illustrated in figure A-10, is input to the F$BGN call that allows the user program to indicate to the operating system the location of the executable binary and a P address which the CPU can be released to. The old BGN format is supported for release 1.14. The following functions are supported with the new BGN format:

- Load a dataset from mass storage as specified by the DSP.
- Copy memory from a source base address to target base address for lengths specified.
- Preset memory with supplied pattern from preset base address for lengths specified.

Support is included for the separation of instruction and data segments. Instruction segments are currently supported and any attempt to load a data segment will be aborted.

Define the F$BGN Function codes:

BGNLOAD	=	O'1	Load from dataset function code
BGNCOPY	=	O'2	Copy from source to destination
BGNFMAX	=	BGNCOPY	Set max Function Code value

```
     0....+....1....+....2....+....3....+....4....+....5....+....6...

     +++-------------+-----------------------+-------+---------------+
   0 |||/////////////|         PRGL          |///////|      FC       |
     +++-------------+-----------------------+-------+---------------+
   1 |||                             PSV                             |
     +++-------------------------------+-----+-----------------------+
   2 |||              PAD              |/////|          ENT          |
     +++-------------------------------+-----+-----------------------+
   3 |///////////////////////////////////////////////////////////////|
     +                                       +-----------------------+
   4 |///////////////////////////////////////|          DSP          |
     +-------------------------------+-------+-----------------------+
   5 |              IBA              |              IBL              |
     +-------------------------------+-------------------------------+
   6 |              DBA              |              DBL              |
     +-------------------------------+-------------------------------+
   7 |             IHLM              |             DHLM              |
     +-------------------------------+-------------------------------+
  10 |             PDBA              |             PDBL              |
     +-------------------------------+-------------------------------+
  11 |             SIBA              |             SIBL              |
     +-------------------------------+-------------------------------+
  12 |             SDBA              |             SDBL              |
     +-------------------------------+-------------------------------+
```

Figure A-10. Begin Code Execution Table (BGN)

Field	Word(base8)	Bits	Description
BGPSF	0	0	Preset value flag,If=1,preset segment
BGEMA	0	1	EMA setting for new calls, 1=ENABLE
BGPRGL	0	16-39	Program length(Old BGN Format only)
BGFC	0	48-63	BGN Function Code(0 for old)
BGPSV	1	0-63	Preset value
BGBP	2	0	Breakpoint flag
BGNRD	2	1	No reduce bit
BGPAD	2	2-33	Pad value
BGENT	2	40-63	Entry point for instruction segment
BGDSP	4	40-63	DSP address of load dataset
BGIBA	5	0-31	Instruction base address to load to
BGIBL	5	32-63	Instruction segment length
BGDBA	6	0-31	Data base address to load to
BGDBL	6	32-63	Data segment length
BGIHLM	7	0-31	Instruction segment HLM value
BGDHLM	7	32-63	Data segment HLM value
BGPDBA	10	0-31	Preset data base address for pattern
BGPDBL	10	32-63	Preset data length for pattern
BGSIBA	11	0-31	Source Instruction base address(COPY)
BGSIBL	11	32-63	Source Instruction length(COPY)
BGSDBA	12	0-31	Source Data base address(COPY)
BGSDBL	12	32-63	Source Data length(COPY)

DATASET DEFINITION LIST - DDL

A Dataset Definition List in the user field must accompany any create DNT (F$DNT) request. The DDL is illustrated in figure A-11.

```
     0....+....1....+....2....+....3....+....4....+....5....+....6...

     +-------------------------------------------------------+-------+
   0 |                          DN                           |///////|
     +-------------------------------------------------------+-------+
   1 |                              LDV                              |
     +++++++---------++-----------+-----------+----------------------+
   2 |||||||| BFI    || ERC       |///////////|          SZ          |
     +++++++---------++-----------+-----------+-------++-------------+
   3 |||//////////////||          DNT         |///////||      BFZ    |
     +++-------------++++---------+-----------+-------++-------------+
   4 |////////////////||| INC     |///////////////////|      DC      |
     +-----------------+++--------+-----------+-------+--------------+
   5 |////////////////////////////////////////|          LM          |
     +----------------------------------------+----------------------+
```

Figure A-11. Dataset Definition List (DDL)

Field	Word(base8)	Bits	Description
DDDN	0	0-55	Dataset name
DDLDV	1	0-63	Logical device name
DDRDM	2	0	Random dataset flag: 0 Sequential 1 Random
DDUDS	2	1	Undefined dataset structure: 0 COS blocked dataset structure 1 Undefined structure
DDNFE	2	2	Return error if dataset does not exist. Register S0 returned nonzero if DNT does not exist; no DNT is created.
DDSTAT	2	3	Request dataset statistics; ignored unless DDNFE=1 (see DDDNT)
DDMR	2	4	Dataset is to be memory resident
DDIA	2	5	Interactive type dataset

Field	Word(base8)	Bits	Description
DDTRAN	2	6	Transparent mode for interactive dataset
DDBFI	2	7-15	Blank field indicator (octal) for character I/O: Value Indicator =000 BFI=I@BFI 400 BFI=User-specified ASCII character =400 BFI=000 >400 Blank compression disabled
DDNA	2	16	No-Abort flag
DDERC	2	17-27	Error code if No-Abort set
DDSZ	2	40-63	Dataset size in 512-word blocks
DDSEQ	3	0	Change a dataset from random to sequential. Valid only if dataset is currently random, ignored if sequential
DDBLK	3	1	Change a dataset form unblocked to blocked. Valid only if dataset is currently unblocked, ignored if blocked
DDDNT	3	16-39	Address of DNT image returned by F$DNT when DDNFE=1 and DDSTAT=1
DDNOF	3	48	No Overflow flag
DDBFZ	3	49-63	Buffer size in 512-word blocks $SYSTXT name
DDC	4	17	Allocate contiguous space for request
DDINC	4	19-27	Sectors to allocate per request
DDDC	4	48-63	Disposition code (two characters):
DDLM	5	40-63	Dataset size limit in 512-word blocks

OPEN DATASET NAME TABLE - ODN

A 2-word Open Dataset Name Table (ODN) is generated in the user field the first time an OPEN of the specified dataset is encountered. Figure A-12 illustrates the ODN.

```
     0....+....1....+....2....+....3....+....4....+....5....+....6...

     +--------------------------------------------------------+-------+
   0 |                          DN                            |///////|
     +++++++-+-----------------------+---+---+----------------+-------+
   1 |||||||/|           LDT          |///|OST|          DSP           |
     +++++++-+-----------------------+---+---+------------------------+
```

Figure A-12. Open Dataset Name Table (ODN)

Field	Word(base8)	Bits	Description
ODDN	0	0-55	Dataset name
ODV	1	1	Close volume
ODM	1	2	Open for 'mod' (append)
ODS	1	3	Close or open with saved position
ODH	1	4	Hold resources
ODUDS	1	5	Open as unblocked flag
ODLDT	1	8-31	LDT address
ODOST	1	36-39	Type of open requested OSTSA=0 Create DSP/LFT buffer in system area OSTUA=1 Create DSP/LFT/buffer in user area OSTMSY=2 DSP/LFT/buffer moved to system area
ODDSP	1	40-63	DSP pointer: Negative: negative offset Positive: absolute address

OPTION TABLE - OPT

The Option Table (OPT) is used for F$OPT calls. Figure A-13 illustrates the OPT.

```
    0....+....1....+....2....+....3....+....4....+....5....+....6...

    +----------------------------------------------------------------+
  0 |                              LPP                               |
    ++++---+---------------------------------------------------------+
  1 ||||PNN|/////////////////////////////////////////////////////////|
    ++++---+---------------------------------------------------------+
```

Figure A-13. Option Table (OPT)

Field	Word(base8)	Bits	Description
OPLPP	0	0-63	Page length
OPSTAT	1	0	Dataset statistics enabled
OPPNCH	1	1	NZ if OPTION,PN selected
OPPNAS	1	2	NZ if PN=n, ZR if PN=ANY
OPPNN	1	3-6	Processor number (if @OPPNAS NZ)

JCL BLOCK INFORMATION TABLE - JBI

The 1-word JCL Block Information Table (JBI) is generated in the user field and has two formats: one for conditional information (see figure A-14) and the other for interative information (see figure A-15).

Conditional block information:

```
     0....+....1....+....2....+....3....+....4....+....5....+....6...

     ++--------------+---------------+---------------+---------------+
   0 ||//////////////|      LLEV     |      PLEV     |      LEVL     |
     ++--------------+---------------+---------------+---------------+
```

Figure A-14. JCL conditional block information

Field	Word(base8)	Bits	Description
JBEXC	0	0	Conditional sequence is in execution
JBLLEV	0	16-31	Conditional is contained in this iterative nesting level
JBPLEV	0	32-47	Iterative is contained in this procedure level
JBLEVL	0	48-63	Current iterative nesting level

Iterative block information:

```
     0....+....1....+....2....+....3....+....4....+....5....+....6...

     +-------+-----------------------+---------------+---------------+
   0 |///////|          CNT          |      PLEV     |      LEVL     |
     +-------+-----------------------+---------------+---------------+
```

Figure A-15. JCL iterative block information

Field	Word(base8)	Bits	Description
JBCNT	0	8-31	Iteration count
JBPLEV	0	32-47	Iterative is contained in this procedure level
JBLEVL	0	48-63	Current iterative nesting level

JCL SYMBOL TABLE - JST

The 4-word JCL Symbol Table (JST) is generated in the job table area and contains information about system and user symbols. See figure A-16.

```
      0....+....1....+....2....+....3....+....4....+....5....+....6...

      ++----------------------------------------------------------------+
    0 ||////////////////////////////////////////////////////////////////|
      ++----------------------------------------------------------------+
    1 ||                           SN                                    |
      ++++++----+-----+-----------------------+-----------------------+
    2 |||||////|TYPE |////////////////////////|          LEVL          |
      ++++++----+-+---+-------------------+---+-----------------------+
    3 |///////////|          LEN           |////|          VAL           |
      +-----------+------------------------+---+-----------------------+
```

Figure A-16. JCL Symbol Table (JST)

Field	Word(base8)	Bits	Description
JSCRE	0	0	Create if not found. Available only for system use.
JSSN	1	0-63	Symbol name
JSLOC	2	0	Local or global. If set, symbol is procedure local.
JSCON	2	1	Constant or variable. If set, symbol is constant.
JSSRS	2	2	System reserved. If set, the symbol name is reserved by the system.
JSUSR	2	3	User settable. If set, symbol may be modified by the job.
JSSYS	2	4	System settable. If set, the symbol may be modified by COS.
JSTYPE	2	10-15	One of the following symbol types: SYMTUND=O'00 Undefined - no type SYMTBOO=O'01 Boolean - logical SYMTINT=O'02 Decimal integer SYMTLIT=O'03 SCII literal; 1-8 characters.

Field	Word(base8)	Bits	Description
JSLEVL	2	40-63	Procedure definition level
JSLEN	3	12-35	Length of value
JSVAL	3	40-63	Base of value buffer

LABEL DEFINITION TABLE - LDT

The Label Definition Table describes the tape label. It consists of four parts: the LDT header, volume header, header 1 entry, and header 2 entry. Except for the LDT header, which points to the other entries, these entries are optional and can appear anywhere after the header. The following conditions must be met for constructing a Label Definition Table (LDT):

- The header must be present.
- The header must precede the first entry.
- Each entry must be pointed to by the offset value in the LDT header. Zero is used for absent fields.
- The lengths of the whole LDT and of each entry must be set in the proper fields.
- The length value for volume 1 must be at least large enough to include the first VSN. The length value for either header 1 or header 2 must be at least the defined length of the respective entry.

LDT HEADER

The LDT header, required on all LDTs, serves the following functions:

- Specifies the beginning and end of the LDT
- Specifies the location of each LDT entry with respect to the LDT base
- Identifies non-standard aspects of a dataset
- Points to labels within a label group

The LDT header is illustrated in figure A-17.

```
    0....+....1....+....2....+....3....+....4....+....5....+....6...

    +------------------------+------------------------+---------------+
  0 |           TN           |////////////////////////|       TL      |
    +---+---++++-------------+---+---+--------+-------+---------------+
  1 |FD |LT ||||/////////////////|IDC|////////|          DNT          |
    +---+---++++----+------------+---+--------+-----------------------+
  2 |///////////////|          V1B            |          H1B          |
    +---------------+-------------------------+-----------------------+
  3 |///////////////|          H2B            |///////////////////////|
    +---------------+-------------------------+-----------------------+
```

Figure A-17. Label Definition Table (LDT) header

Field	Word(base8)	Bits	Description
LDTN	0	0-23	Table name ('LDT' in ASCII)
LDTL	0	48-63	Table length (variable)
LDFD	1	0-3	Foreign dataset translation identifier. This field is used to indicate whether run time foreign dataset translation should be performed on this dataset.
LDLT	1	4-7	Requested label type: 0 TPLNL Non-labeled 1 TPLAL ANSI-standard label 2 TPLSL IBM standard labels
LDPROT	1	8	Protected access indicator. If non-zero for a new tape dataset then the dataset is to be protected on the servicing front-end.
LDCAT	1	9	Cataloged dataset indicator
LDCV	1	10	Dataset data conversion flag. This field is used to indicate whether implicit data conversion shall be done by the run time library.
LDIDC	1	28-31	Initial dataset desposition 0 TPOLD Old dataset 1 TPNEW New dataset

LDDNT	1	40-63	Dataset name table (DNT) pointer. The field value is JTA-relative.
LDV1B	2	16-39	Offset of volume 1 entry, relative to LDT base. If the LDT does not contain a VOL1 entry, this field must be zero.
LDH1B	2	40-63	Offset of header 1 entry, relative to LDT base; must be zero if there is no HDR1 entry.
LDH2B	3	16-39	Offset of header 2 entry, relative to LDT base; must be zero if there is no HDR2 entry.

If the tape label is being sent to the front-end for servicing, word 1 of the Label Definition Table (LDT) header is redifined as shown in figure A-18.

```
    0....+....1....+....2....+....3....+....4....+....5....+....6...

    +----------------------------------------+-----------------------+
  1 |////////////////////////////////////////|          FSH          |
    +----------------------------------------+-----------------------+
```

Figure A-18. Header redefiniton of LDDNT

Field	Word(base8)	Bits	Description
LDFSH	1	40-63	Front-end service header offset

VOLUME 1 ENTRY

The volume 1 entry (see figure A-19) corresponds to volume 1 labels for all volumes in the dataset. The volume 1 entry may be placed anywhere after the header, so long as the LDV1B header field points to it properly. The volume 1 entry is optional.

```
     0....+....1....+....2....+....3....+....4....+....5....+....6...

     +-------------------------------+---------------+---------------+
   0 |             VOL1              |      VSB      |     VL1L      |
     +---------------+---------------+-------+-------+---------------+
   1 |     VSNL      |      CVN      |  DT   |///////|      FVN      |
     +---------------+---------------+-------+-------+---------------+
   2 |                              OID                              |
     +                                               +---------------+
   3 |                                               |///////////////|
     +-----------------------------------------------+---------------+
   4 |                              GDN                              |
     +-----------------------------------------------++--+---+-------+
   5 |                      VSN                      ||//|VDC|  VAC  |
     +-----------------------------------------------++--+---+-------+
```

Figure A-19. Label Definition Table (LDT) volume 1 entry

Field	Word(base8)	Bits	Description
LDVOL1	0	0-31	Entry name ('VOL1' in ASCII)
LDVSB	0	32-47	Volume serial list base offset
LDVL1L	0	48-63	Volume 1 length
LDVSNL	1	0-15	Number of VSNs in entry
LDCVN	1	16-31	Current VSN ordinal
LDDT	1	32-39	Device type 0 TPD62 6250 bpi 1 TPD16 1600 bpi
LDFVN	1	48-63	Final VSN ordinal: ordinal of VSN corresponding with the volume sequence number in access condition
LDOID	2-3	0-63	Owner identifier
LDOID1	2	0-63	Characters 1-8
LDOID2	3	0-47	Characters 9-14
LDGDN	4	0-63	Generic device name
LDVSN	5	0-47	Beginning VSN

Field	Word(base8)	Bits	Description
LDVRG	5	48	Volume-registered flag, set by a servicing front-end. When set, the VSN is from front-end catalog.
LDVDC	5	52-55	Volume disposition 0 TPOLD Existing dataset 1 TPNEW New volume to dataset
LDVAC	5	56-63	Volume accessibility character, obtained from the label group

Word 5 of the Label Definition Table (LDT) volume 1 entry is the beginning of the VSN list. Each VSN requires one word as shown in figure A-20. The maximum number of VSNs is an installation defined parameter, which is given as I@TMV.

```
    0....+....1....+....2....+....3....+....4....+....5....+....6...

    +-----------------------------------------------+----------------+
  5 |                     VSN1                      |////////////////|
    +-----------------------------------------------+----------------+
```

Figure A-20. Beginning of VSN list

Field	Word(base8)	Bits	Description
LDVSN1	5	0-47	

LE@VOL1=W@LDVSN+I@TMV

HEADER 1 ENTRY

The header 1 entry (see figure A-21) describes dataset attributes and corresponds to the HDR1, EOF1, and EOV1 labels for all volumes in the dataset. Header 1 shows numeric fields in both binary and ASCII characters. COS uses the ASCII equivalents for generating and validating the label group. If a field is changed, both versions must be changed. ASCII fields are right-justified with leading zeros. The header 1 entry is optional and can be placed anywhere after the header, so long as it is pointed to by header field LDH1B.

```
     0....+....1....+....2....+....3....+....4....+....5....+....6...

     +-------------------------------+---------------+---------------+
   0 |             HDR1              |///////////////|     HR1L      |
     +-------------------------------+---------------+---------------+
   1 |                             FID1                              |
     +---------------------------------------------------------------+
   2 |                             FID2                              |
     +---------------------------------------------------------------+
   3 |                             FID3                              |
     +---------------------------------------------------------------+
   4 |                             FID4                              |
     +---------------------------------------------------------------+
   5 |                             FID5                              |
     +-------------------------------+---------------+---------------+
   6 |             FID6              |     CVSQ      |     FVSQ      |
     +-------------------------------+---------------+---------------+
   7 |             FSEC              |             CSEC              |
     +-------------------------------+-------+-------+---------------+
  10 |             FSEQ              |  DAC  |  VN   |      FSQ      |
     +-------------------------------+-------+-------+---------------+
  11 |              GEN              |      GN       |      GVN      |
     +-------------------------------+---------------+---------------+
  12 |                     CDT                       |///////////////|
     +-----------------------------------------------++--------------+
  13 |                     XDT                       ||      RT      |
     +-----------------------------------------------++--------------+
  14 |                     BLK                       |///////////////|
     +-----------------------------------------------+---------------+
  15 |                     SET                       |///////////////|
     +-------------------------------+---------------+---------------+
  16 |              FBC              |              VBC              |
     +-------------------------------+-------------------------------+
  17 |                             SCOD                              |
     +                                       +-----------------------+
  20 |                                       |///////////////////////|
     +---------------------------------------+-----------------------+
```

Figure A-21. Label Definition Table (LDT) header 1 entry

Field	Word(base8)	Bits	Description
LDHDR1	0	0-31	Entry name ('HDR1' in ASCII)
LDHR1L	0	48-63	Header 1 length
LDFID1	1	0-63	Characters 1-8
LDFID2	2	0-63	Characters 9-16
LDFID3	3	0-63	Characters 17-24
LDFID4	4	0-63	Characters 25-32
LDFID5	5	0-63	Characters 33-40
LDFID6	6	0-31	Characters 41-44
LDCVSQ	6	32-47	Current volume sequence number (file section number), binary equivalent of LDCSEC
LDFVSQ	6	48-63	First volume sequence number (file section number), binary equivalent of LDFSEC
LDFSEC	7	0-31	First file section number (volume sequence number) in ASCII, the ordinal number of the volume to be mounted first
LDCSEC	7	32-63	Current file section number (volume sequence number) in ASCII, the ordinal number of the currently mounted volume
LDFSEQ	10	0-31	File sequence number (ASCII) ordinal of the dataset being accessed. If FSEQ < 1, volume should have more than one dataset.
LDDAC	10	32-39	Dataset accessibility character.
LDVN	10	40-47	Generation version number, numeric equivalent of LDGVN
LDFSQ	10	48-63	File sequence number, numeric equivalent of LDFSEQ

Field	Word(base8)	Bits	Description
LDGEN	11	0-31	Generation number. Any value other than one indicates that a dataset is in a generation data group.
LDGN	11	32-47	Generation number, numeric equivalent of LDGEN
LDGVN	11	48-63	Generation version number (ASCII). Any value other than 0 indicates that the dataset is in a generation data group.
LDCDT	12	0-47	Creation date (ASCII). This field indicates the creation date of the dataset in the julian form: 'yyddd'. Note the space (LDCSP) must be present.
LDCSP	12	0-7	Space
LDCYR	12	8-23	Year
LDCDY	12	24-47	Day
LDXDT	13	0-47	Expiration date; same format as creation date above
LDXSP	13	0-7	Space
LDXYR	13	8-23	Year
LDXDY	13	24-47	Day
LDUXD	13	48	User specified XDT (expiration date) flag
LDRT	13	49-63	Retention period, integer days
LDBLK	14	0-47	Volume block count (ASCII): number of user data blocks present, read from or written into the label. Can be inaccurate because overflow causes it to be cleared; see LDVBC for an accurate count.
LDSET	15	0-47	File set identifier, normally set to the serial number of first volume in the dataset
LDFBC	16	0-31	File block count (binary)

Field	Word(base8)	Bits	Description
LDVBC	16	32-63	Volume block count (binary), number of blocks written on volume so far
LDSCOD	17-20	0-63	System identification code, to identify the operating system or computer system that generated the tape
LDSCD1	17	0-63	Character 1-8
LDSCD2	20	0-39	Character 9-13 Identify the operating system or computer system that generated the tape: LE@HDR1=W@LDSCD2+1

HEADER 2 ENTRY

The header 2 entry (see figure A-22) describes dataset attributes and corresponds to the HDR2, EOF2, and EOV2 labels for all volumes in the dataset. Header 2 shows numeric fields in both binary and ASCII characters. COS uses the ASCII equivalents for generating and validating the label group. If a field is changed, both versions must be changed. ASCII fields are right-justified with leading zeros. The header 2 entry is optional and can be placed anywhere after the header, as it is pointed to by header field LDH2B.

```
    0....+....1....+....2....+....3....+....4....+....5....+....6...

    +-------------------------------+---------------+---------------+
  0 |             HDR2              |///////////////|     HR2L      |
    +-------+-------+-------+-------+---------------+---------------+
  1 |  FMT  |  BA   |  RF   |///////|              MBS              |
    +-------+-------+-------+-------+-------------------------------+
  2 |      BFO      |///////////////|              MRS              |
    +---------------+---------------+-------+-----------------------+
  3 |                  BL                   |///////////////////////|
    +---------------------------------------+-----------------------+
  4 |                  RL                   |///////////////////////|
    +---------------------------------------+-----------------------+
```

Figure A-22. Label Definition Table (LDT) header 2 entry

Field	Word(base8)	Bits	Description
LDHDR2	0	0-31	Entry name ('HDR2' in ASCII)
LDHR2L	0	48-63	Header 2 length
LDFMT	1	0-7	Record format, two types IBM label types: F Fixed-length records V Variable-length records U Undefined record format ANSI label types: F Fixed-length records D Variable-length records S Records span tape blocks
LDBA	1	8-15	Blocking attributes, IBM label types only: B Blocks are an integral multiple of the record size S Records span tape blocks R Records span tape blocks, and the blocks are an integral multiple of the record size
LDRF	1	16-22	Record format.
LDMBS	1	32-63	Maximum block size (binary), maximum size of any tape block that can be read or written
LDBFO	2	0-15	Buffer offset, ANSI only (not currently supported by COS)
LDMRS	2	32-63	Maximum record size (binary), maximum size of any record that can be read or written
LDBL	3	0-39	Maximum block size (ASCII), maximum number of bytes in a tape block, read from or written into the label. Can be inaccurate because overflow causes it to be cleared; see LDMBS for an accurate count.
LDRL	4	0-39	Maximum record size (ASCII), maximum number of bytes in a tape record, read from or written into the label. Can be inaccurate because overflow causes it to be cleared; see LDMRS for an accurate count.

LE@HDR2=W@LDRL+1

EVENT RECALL PARAMETER BLOCK - ERPB

The ERPB, shown in figure A-23, is the parameter block used with F$ERCL requests.

```
     0....+....1....+....2....+....3....+....4....+....5....+....6...

     +-------+---------------------------------+----------------------+
   0 |ERFUNC|/////////////////////////////////|        ERTO          |
     +------+--------+------------------------+----------------------+
   1 |////////////////|         ERMASK         |        ERMAP         |
     +----------------+------------------------+----------------------+
```

Figure A-23. Event Recall parameter block

Field	Word(base8)	Bits	Description
ERFUNC	0	0-6	Subfunction code
			The functions range from ERCL$$MI to ERCL$$MA-1. When subfunctions are added adjust the ERCL$$ symbols as needed.
			ERCL$DIS=01 Disable event monitoring
			ERCL$ENA=02 Enable event monitoring
			ERCL$RCL=03 Recall untill event
			ERCL$RET=04 Return occurred-events map
			ERCL$$MI=01 Minimum subfunction
			ERCL$$MA=05 Maximum subfunction+1
ERTO	0	40-63	Timeout value (milliseconds)
ERMASK	1	16-39	Event selection mask
			ERCL$$ values must be changed when new events are added. Bits zero thru ERM$$MAX-1 must always be defined. Bits ERM$$FP thru ERM$$LP-1 must always be defined.
ERMSIJ	1	16	Inter-job message arrived
ERMSUO	1	17	Unsolicited oper msg arrived

Field	Word(base8)	Bits	Description
ERMSOR	1	18	Operator reply arrived
			ERM$$MAX=D'18+1 Last non-privileged bit+1
			ERM$$FP=D'26 First privileged bit
ERMSCH	1	26	Channel function done
ERMSIQ	1	27	SDT placed in INPUT queue
ERMSOQ	1	28	SDT placed in OUTPUT queue
			ERM$$LP=D'28+1 Last privileged bit+1
ERMAP	1	40-63	Occurred-events map
ERMPIJ	1	40	Inter-job message arrived
ERMPUO	1	41	Unsolicited oper msg arrived
ERMPOR	1	42	Operator reply arrived
ERMPCH	1	50	Channel function done
ERMPIQ	1	51	SDT placed in INPUT queue
ERMPOQ	1	52	SDT placed in OUTPUT queue
			On return from F$ERCL, S0 can have the following values.
		00	Okay
			ERER$MT=01 Prohibited to multitasking job
			ERER$PV=02 Not a privileged job
			ERER$BFN=03 Bad function
			ERER$UDB=04 Mask contains undefined bits
			ERER$MDI=05 Monitoring not enabled

USER DRIVER PARAMETER BLOCK - DRPB

The DRPB, as shown in figure A-24, is the parameter block used with F$DRIVER requests.

```
     0....+....1....+....2....+....3....+....4....+....5....+....6...

     +---------------------------------------------------------+------+
  0  |/////////////////////////////////////////////////////////|PLEN  |
     +---------------------------------------------------------+------+
  1  |                            FUNC                                |
     +----------------------------------------------------------------+
  2  |                            COSS                                |
     ++---------------------------------------------------------------+
  3  ||///////////////////////////////////////////////////////////////|
     ++---------------------------------------------------------------+
  4  |                             DRS                                |
     +----------------------------------------------------------------+
  5  |                             DP                                 |
     +----------------------------------------------------------------+
  6  |                             LN                                 |
     +----------------------------------------------------------------+
  7  |                            DRNM                                |
     +------------------------------------------------+---------------+
 10  |////////////////////////////////////////////////|      TO       |
     +------------------------------------------------+---------------+
 11  |                             DIR                                |
     +----------------------------------------------------------------+
 12  |                             OPD                                |
     +----------------------------------------+-----------------------+
 13  |////////////////////////////////////////|        BAD1           |
     +----------------------------------------+-------+---------------+
 14  |////////////////////////////////////////////////|     DLN1      |
     +------------------------------------------------+---------------+
 15  |////////////////////////////////////////////////|     TLN1      |
     +----------------------------------------+-------+---------------+
 16  |////////////////////////////////////////|        BAD2           |
     +----------------------------------------+-------+---------------+
 17  |////////////////////////////////////////////////|     DLN2      |
     +------------------------------------------------+---------------+
 20  |////////////////////////////////////////////////|     TLN2      |
     +------------------------------------------------+---------------+
 21  |                            DFP1                                |
     +----------------------------------------------------------------+
 22  |                            DFP2                                |
     +----------------------------------------------------------------+
 23  |                            DFP3                                |
     +----------------------------------------------------------------+
 24  |                            SPAR                                |
     +----------------------------------------------------------------+
```

Figure A-24. Channel Access parameter block

Field	Word(base8)	Bits	Description
DRPLEN	0	58-63	Parameter block length
DRFUNC	1	0-63	Subfunction code

The following line is a *CALL to comdeck COMAPFC.

CFN$xxx codes are used to specify the type of request to the shell and/or driver.

If codes are added, CFNMIN, CFNRSV, CFN$DMIN, and CFN$DMAX must be updated accordingly.

CFN$MIN=3	Minimum legal code
CFN$OPE=3	Driver Open
CFN$CLS=4	Driver close
CFN$RD=5	Read header
CFN$RDH=6	Read header and hold data
CFN$RDD=7	Read both header and data
CFN$WT=D'8	Write header
CFN$WTH=D'9	Write header and hold data
CFN$WTD=D'10	Write header and data
CFN$RSV=D'11 thru D'31	Reserved
CFN$DMIN=D'32	Minimum legal driver function code
CFN$DMAX=D'127	Maximum legal driver function code

CST$xxx codes are returned by the shell and drivers.

CST$CMP=0	Complete
CST$MIN= CST$CMP	Minimum status
CST$PRO=3	Protocol error
CST$CHN=4	Illegal channel number
CST$FCN=5	Illegal function code
CST$DVN=6	Illegal driver name
CST$DAE=7	Data address error
CST$DLE=D'8	Data length error
CST$MAX= CST$DLE	Maximum status

Field	Word(base8)	Bits	Description
			D'9 - D'31 Reserved CST$DMIN=D'32 Min driver specific code CST$DMAX=D'127 Max driver specific code CST$xxx codes for loopback driver. CST$TMO=D'32 Loopback Driver timeout
DRCOSS	2	0-63	Status of the request. DRS$OK=0 Okay DRS$CNO=1 Channel is not open DRS$CAO=2 Channel is already open DRS$RSV=3 Channel is reserved to another task DRS$CUK=4 Channel is unknown DRS$OFF=5 Channel is off DRS$BA1=6 Bad buffer1 address DRS$BA2=7 Bad buffer2 address DRS$BL1=8 Bad buffer1 length DRS$BL2=9 Bad buffer2 length DRS$BPS=10 Bad parameter size DRS$BDI=11 Bad channel direction DRS$BSY=12 Channel is busy DRS$BFN=13 Bad function DRS$NMT=14 Not available to multitasking jobs DRS$NRS=15 Channel is not reserved to you DRS$MIN= DRS$OK Min status DRS$MAX= DRS$NRS+1 Max status+1
DRCOMS	3	0	'Driver complete' status
DRDRS	4	0-63	Driver and shell status
DRDP	5	0-63	Driver parameter
DRLN	6	0-63	Logical channel name; 1-7 chars. Left justified, blank filled.
DRDRNM	7	0-63	Driver name
DRTO	10	48-63	Driver timeout in tenths of a second

Field	Word(base8)	Bits	Description
DRDIR	11	0-63	Direction of channel DIR$INP=0 Input DIR$OUT=1 Output
DROPD	12	0-63	OPEN driver spare
DRBAD1	13	40-63	Buffer1 address
DRDLN1	14	48-63	Data1 length
DRTLN1	15	48-63	Transfer1 length
DRBAD2	16	40-63	Buffer2 address. Used only with CFNRDD, CFNWTH, and CFN$WTD
DRDLN2	17	48-63	Data2 length. Used only with CFNRDD, CFNRDH, CFN$WTH, and CFN$WTD
DRTLN2	20	48-63	Transfer2 length. Length of data actually transferred from RDBAD2
DRDFP1	21	0-63	DRIVER function parameters
DRDFP2	22	0-63	DRIVER function parameters
DRDFP3	23	0-63	DRIVER function parameters
DRSPAR	24	0-63	Spare for future use

RECEPTIVE CONTROL BLOCK - RCB

The Receptive Control Block (RCB), shown in figure A-25, exists in the user field of jobs that are receptive to inter-job communication requests. There is one RCB per ID.

```
   0....+....1....+....2....+....3....+....4....+....5....+....6...

   +---------------------------------------------------------------+
 0 |                              ID                               |
   +---------------------------------------------------------------+
```

Figure A-25. Receptive control block

Field	Word(base8)	Bits	Description
RCBID	0	0-63	ID of the job requesting connection

NODE CONTROL BLOCK - NCB

The Node Control Block (NCB), shown in figure A-26, exists in the user field of jobs having interjob communication paths and contains information about those paths. There is one NCB per node.

```
     0....+....1....+....2....+....3....+....4....+....5....+....6...

     +----------------------------------------+-----------------------+
   0 |////////////////////////////////////////|         NCPN          |
     +----------------------------------------+-----------------------+
   1 |////////////////////////////////////////|          LEN          |
     +----------------------------------------+-----------------------+
   2 |////////////////////////////////////////|          ADD          |
     +----------------------------------------+-----------------------+
   3 |                               NCST                               |
     +----------------------------------------------------------------+
```

Figure A-26. Node control block

Field	Word(base8)	Bits	Description
NCPN	0	40-63	IPT offset for this path
NCBLEN	1	40-63	Length of the node buffer
NCBADD	2	40-63	Address of the node buffer
NCST	3	0-63	Status
NCMS	3	0	Message status
NCOS	3	48-63	Open status

If any values are changed, SYSLIB must be changed also.

NCB$ACC='AC'R	Open request accepted
NCB$REJ='RJ'R	Open request rejected
NCB$CLO='CL'R	Path was closed

INTERJOB COMMUNICATION MESSAGE BUFFER - MHB

The Interjob Communication Message Buffer, shown in figure A-27, exists in the user field of jobs having interjob communication paths. One buffer per node exists. The buffer contains a header followed by the message body. The buffer header contains information about the message and the buffer body contains the message itself.

```
       0....+....1....+....2....+....3....+....4....+....5....+....6...

       +---------------------------------------------------------------+
     0 |                            MHALEN                             |
       +---------------------------------------------------------------+
     1 |                            MHSLEN                             |
       +---------------------------------------------------------------+
```

Figure A-27. Interjob Communication Message Buffer

Field	Word(base8)	Bits	Description
MHALEN	0	0-63	Length of message put into the buffer
MHSLEN	1	0-63	Length of the message sent
			This header will be followed by message data in the format defined by the communicating tasks.

INTERJOB COMMUNICATION PARAMETER BLOCK - IJPB

The Interjob Communication Parameter Block (IJPB), shown in figure A-28, is the parameter block used with F$IJMSG requests.

```
    0....+....1....+....2....+....3....+....4....+....5....+....6...

    +-------+------+---------------------------------------------------+
 0  | LEN   |******|///////////////////////////////////////////////////|
    +-------+------+                          +------------------------+
 1  |/////////////////////////////////////////|         IJLINK         |
    +-----------------------------------------+------------------------+
 2  |                              IJSTAT                              |
    +------------------------------------------------------------------+
 3  |                              IJFUNC                              |
    +------------------------------------------------------------------+
 4  |                              IJRID                               |
    +------------------------------------------------------------------+
 5  |                              IJTID                               |
    +-----------------------------------------+------------------------+
 6  |/////////////////////////////////////////|          IJRCB         |
    +-----------------------------------------+------------------------+
 7  |/////////////////////////////////////////|          IJNCB         |
    +-----------------------------------------+--------+---------------+
10  |//////////////////////////////////////////////////|    IJBLEN     |
    +-----------------------------------------+--------+---------------+
11  |/////////////////////////////////////////|         IJBADD         |
    +-----------------------------------------+------------------------+
12  |                              IJOVR                               |
    +------------------------------------------------------------------+
13  |                              IJFCS                               |
    +------------------------------------------------------------------+
14  |                              IJFCU                               |
    +------------------------------------------------------------------+
15  |                              IJCLS                               |
    +------------------------------------------------------------------+
```

Figure A-28. Interjob communication parameter block

Field	Word(base8)	Bits	Description
IJPLEN	0	0-6	Length of the parameter block
IJHLEN	0	7-12	Message buffer header length (LH@MHB)
IJLINK	1	40-63	Link to next parameter block
IJSTAT	2	0-63	Status IJMS$OK=00 Completed with no error The following responses do not terminate a request chain. If any values are changed, SYSLIB must be changed also.

Field	Word(base8)	Bits	Description
			IJMS$AR=01 ID is already receptive IJMS$AU=02 ID is in use IJMS$BA=03 Buffer address or length bad IJMS$BN=04 NCB is bad IJMS$BNA=05 NCB address is bad IJMS$BP=06 Path is busy IJMS$HL=07 HLEN error IJMS$IF=08 IPT full IJMS$INR=09 ID not registered IJMS$INS=10 ID not specified IJMS$MC=11 Bad log message class IJMS$ML=12 Bad message length IJMS$NA=13 ID is not attached IJMS$NE=14 Path is not open IJMS$NO=15 No outstanding open request IJMS$NP=16 Path does not exist IJMS$NR=17 ID is not receptive IJMS$OO=18 Outstanding OPEN was found IJMS$PE=19 Path is already established IJMS$PF=20 Memory pool is full IJMS$PR=21 ID is privileged IJMS$RB=22 Bad RCB address IJMS$RF=23 RIT full IJMS$TA=24 Target's buffer address is bad IJMS$TL=25 Target's buffer length is bad
			The following responses terminate a request chain.
			IJMS$BE=32 IJPB length error IJMS$BF=33 Undefined function IJMS$LA=34 Bad link address IJMS$MT=35 More than one active TXT IJMS$NC=36 RIT or IPT has zero entries IJMS$PV=37 Privileged function IJMS$TP=38 More than I@MPBS parameter blocks IJMS$MAX=39 Maximum status value + 1
IJFUNC	3	0-63	Subfunction code If any values are changed, SYSLIB must be changed also.

Field	Word(base8)	Bits	Description
			IJM$NOP=00 No op
			IJM$REC=01 Request receptivity state
			IJM$OPEN=02 Open a communication path
			IJM$ACCE=03 Accept an IJM$OPEN request
			IJM$REJE=04 Reject an IJM$OPEN request
			IJM$SNDM=05 Send a message
			IJM$CLOS=06 Close a communication path
			IJM$END=07 Ends the receptivity state
			IJM$$HNP=07+1 Maximum value + 1 of unprivileged subfunctions
			IJM$$MIP=32 Minimum privileged function value
			IJM$SNDL=32 Send a logfile message (privileged)
			IJM$$MAX=32+1 Maximum subfunction value + 1
IJRID	4	0-63	ID of the requesting job
IJTID	5	0-63	ID of the target job
IJRCB	6	40-63	RCB address
IJNCB	7	40-63	NCB address
IJBLEN	10	48-63	Message buffer length
IJBADD	11	40-63	Message buffer address
IJOVR	12	0-63	Log message over-ride flag
IJFCS	13	0-63	Log message to system log
IJFCU	14	0-63	Log message to user log
IJCLS	15	0-63	Log message class

SUBSYSTEM SUPPORT B

Subsystem Support provides a mechanism for the user to develop code that would otherwise have to be incorporated as part of the Cray Operating System (COS). Examples of this kind of code are networking packages and online diagnostics.

Subsystem Support is a collection of independent functions whose use may be restricted to jobs granted the necessary privilege by COS.

This appendix describes the following Subsystem Support functions:

- Interjob communication
- User channel access
- Event recall

INTERJOB COMMUNICATION

A job may communicate with one or more other jobs. This feature is available to all single-tasking jobs. It is prohibited to multitasking jobs.

To establish communication, one job indicates it is receptive to communication, and the others request to open a communication path between themselves and the receptive job. Once a path is established, jobs may freely exchange messages. Any one job may open as many communication paths as it needs. An installation defined parameter, I@MIJPA, determines the total number of communication paths allowed in the system at one time.

Message exchange is memory to memory between jobs if both are resident. Otherwise, messages are queued for rolled out jobs. The maximum length of a message is determined by an installation defined parameter, I@MIJML.

A receptive job may place a message in the user logfile of any connected job. This is a privileged function.

ESTABLISHING COMMUNICATION

Each job must have at least one unique nonzero 64-bit ID. The programmer chooses the ID, so must know the IDs of the communicating jobs. See your site analyst for the IDs of system supported programs. Because system supported programs commonly have IDs that begin with a $, refrain from using this format when choosing an ID.

A job becomes receptive through a system request specifying its ID and the location of its Receptive Control Block (RCB). The system uses this RCB when processing requests from other jobs to find out if this job allows a communication path to be established with another job. The RCB is one word long and is set to 0 by the system when the job becomes receptive. When another job makes a request to open communication, that job's ID is placed in the RCB. The RCB is always set by the system and read by the user. The user should never write into the RCB.

A job attempts to establish a communication path with another job by making a system request, specifying its ID and the ID of the target job. If the target job is receptive the system will put the requesting job's ID into the target job's RCB if the target job is resident and its RCB is 0. Otherwise, the request is queued. No further requests may be made to the target job until a response is received. The target job polls its RCB for a nonzero value, indicating a request for connection. The target job screens out undesirable jobs. The target job accepts or rejects the attempt to establish communication by making a system request and indicating its response (accept or reject), its ID, and the ID of the initiating job. If it accepts, the communication path is established, and messages can be transferred freely. The job that requested that a communication path be established is said to be attached to the target job. Upon receipt of the response, whether accept or reject, the system places in the RCB, the ID of the next job requesting that a path be established. If there is no job requesting a path, the RCB is set to 0.

The communication path consists of two nodes, one in each job. Each node consists of a Node Control Block (NCB) and a message buffer (MHB). The NCB consists of a pointer to the message buffer, the length of the buffer, and status indicators. The message status indicator must be polled to see if a message arrived. A zero-length message indicates a change in open status may have occurred. The open status indicates whether a reply to an open request arrived or the other job closed this path. Each job is responsible for clearing its message status after it has taken appropriate action. No further messages will be put into a buffer until the message status is 0. The NCB allows dynamic message buffers. The job may change the size of MHB with or without relocating it, but this change can only be made when the message status has been set by the system. When the message status has been set by the system, the system will not do anything further with that buffer until the user clears the status. The job can then change the buffer and clear the message status (in that order) so the system resumes message transfer to that node.

When a job requests that a communication path be established with another job, the requesting job sends the location of its NCB in the request. When the target job replies, it sends the location of its NCB in the reply. So, the communication path is well defined. Figure B-1 illustrates a typical subsystem interjob communication structure.

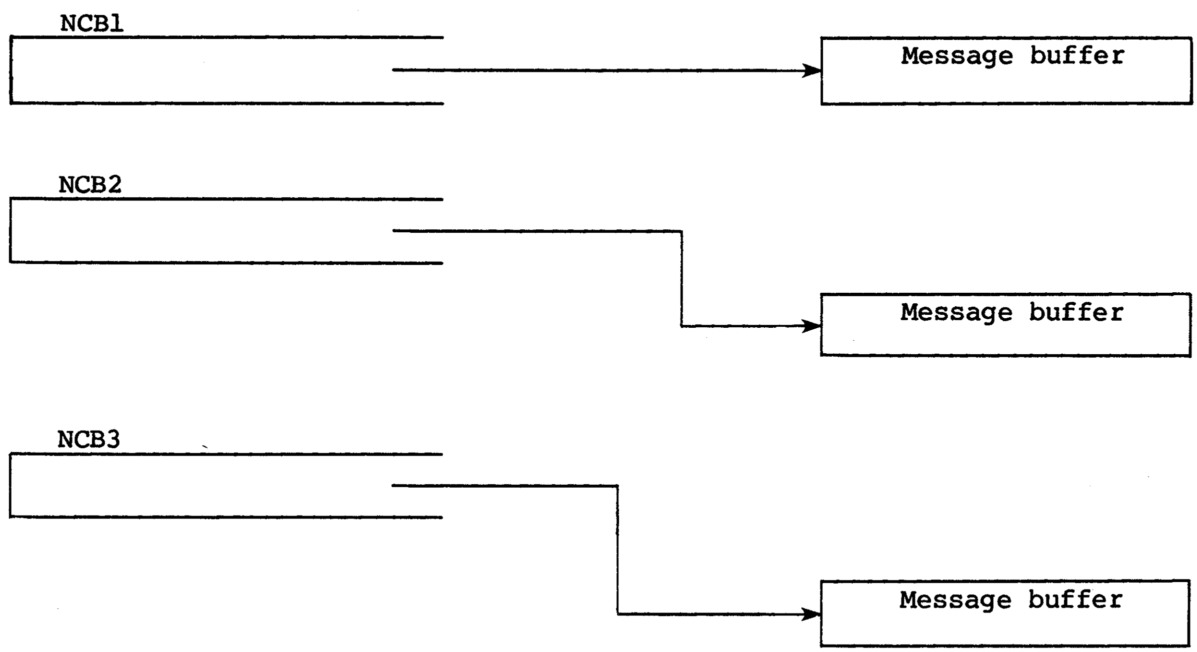

Figure B-1. A typical subsystem interjob communication structure

The job in figure B-1 has communication paths established with three other jobs. Messages from JOB1 are placed in the buffer pointed to by NCB1, while JOB3's messages are placed in the buffer pointed to by NCB3. The location of the buffers is not important. The NCBs, however, should be allocated so V registers can be used to poll for nonzero status values.

A job may use more than one ID in its communications. This allows for multiple paths between jobs.

SENDING AND RECEIVING MESSAGES

When a communication path has been established, a job sends a message by making a system request indicating the location and length of the message to be sent, and an NCB address.

If a job's NCB message status indicates that a message has been placed in its message buffer, the job may read the message in the buffer and clear its NCB message status. The message appears directly after a message header. The header contains the length of the message sent and the length of the message actually put in the buffer. A message that is too large for the buffer is truncated. No further action is taken by the system.

Message exchange is memory to memory when both jobs are resident. Otherwise, one message per node is queued for any job that is rolled out or has a nonzero NCB message status. All requests to send a message to a job that already has a message queued are rejected with a busy status. If no pool space is available to queue a message, a pool-full message is returned. The job tries again later.

When a program removes messages from message buffers and clears the message status, it issues an event recall return or recall function. This assures that queued messages move into the buffers as quickly as possible rather than wait for the system to detect that buffers are available for new messages.

Sending an ASCII message to an attached job's logfile is a privilege and can be done by making a system request specifying the location of the message, an NCB address, a message class indicator, destination indicator, and an Override flag. The message must be 1 through 80 characters and must be terminated by a zero byte if less than 80 characters.

CLOSING COMMUNICATION PATHS

A job may close *all* communication paths with a given ID by specifying that ID and an NCB address. A job closes a *specific* communication path by making a system request specifying its ID, an NCB address, and another job's ID. The closing job informs the other job of its intention to close communication before the close request is made. Any messages queued on either end of this path are discarded and a zero-length "path closed" message is placed in the other job's message buffer or queued for the other job. If a job receives a zero-length message it checks its NCB open status for a change.

A job gives up its receptivity by making a system request specifying its ID. This request does not affect existing communication paths but prevents future open requests' referring to that ID from being posted. If there are any open requests pending when this request is made, a status indicator is returned in the NCB, and the ID is placed in the RCB. The job's receptivity is ended, but the job continues to accept or reject open requests until the RCB is returned with a 0 value. The 0 indicates that no more open requests are queued for this job. If the job does not perform this function, the queued open requests remain until either the job becomes receptive again or job advance occurs.

All communication must be closed before the end of each job step or the job aborts. Communication paths do not affect the recoverability of a job. If a job with paths established is recovered, all paths are eliminated and the job reestablishes the paths. A job using an established communication path detects this occurrence when an ID-not-established status is returned in response to a communication request.

SYSTEM REQUESTS

The system requests available are F$IJMSG requests with the following functions: IJMNOP, IJMREC, IJM$OPEN, IJM$ACCE, IJM$REJE, IJM$SNDM, IJM$SNDL, IJM$CLOS, and IJM$END. Each request requires a parameter block (IJPB). Up to an installation-defined maximum number of parameter blocks (I@MPBS) can be linked together allowing for multiple requests with one F$IJMSG system request.

USER CHANNEL ACCESS

A job can communicate directly with a user-supplied driver using open, read, write, close, and special driver requests. These requests require the specification of a logical channel name, a return status word, and various buffer information. This is a privileged feature available to single-tasking jobs but prohibited to multitasking jobs.

A user accesses a user-supplied driver with the F$DRIVER system request, DRIVER macro, or DRIVER FORTRAN subroutine. Only one request for a channel may be outstanding at a time.

The user opens a channel by specifying a logical channel name, a channel timeout value, a driver name, and an I/O direction. If no timeout value is specified, the system uses an installation-defined value (I@CHATIM). All subsequent functions on this channel use this value unless a timeout value is specified with a specific function. Specify the driver name only if the system is *not* to use the standard driver for the given channel. The input (or output) channel must be opened before it can be read (or written). Opening the channel automatically reserves it. The system rejects all subsequent requests from other jobs for that channel until the job closes the channel.

Close a channel by specifying the channel name and direction. The channel reservation is released when the channel is closed.

The user can send a message to the operator requesting that a channel be turned on or off (see Job-to-Operator Communication).†

Transfer data by specifying the channel name (the direction is not needed), the address of the buffer to or from which data is to be transferred, and the length of the data to be transferred. The system returns the length of the data actually transferred.

Issue special requests defined in the individual driver specifications by specifying the channel name and direction. See the individual driver specifications for other requirements.

For each function, send additional data to the driver (for example, the timeout value for this function) in a reserved driver word in the request parameter block. The driver returns information to the user in this word. See the individual driver specifications for the use of this word.

Job termination closes and releases all channels currently belonging to a job. Open channels do not affect the recoverability of a job. In the event that a job with opened channels is recovered, all channel links are eliminated and it is the responsibility of the job to re-open them. A job can detect this occurrence when the status "Channel doesn't belong to you" is returned in response to a channel request.

EVENT RECALL

An event recall request can cause a job to suspend until an event occurs. When the event occurs, the job is recalled and the event reported. This feature is available to all single-tasking jobs but is prohibited to multitasking jobs.

Event recall has two phases: waiting for events and discovering whether events have occurred. If one or more of the following events is requested in a job, the job is released from recall when the event occurs. Note that some of the events are privileged.

- Timeout elapsed
- Interjob communication message received
- Unsolicited operator message received†
- Operator reply received†
- Channel driver completed (privileged)
- An SDT placed in the INPUT queue (privileged)†
- An SDT placed in the OUTPUT queue (privileged)†

---

† Deferred implementation

A timeout event is always enabled in order to prevent a job's being suspended indefinitely.

The F$ERCL system request, ERECALL system macro, and FORTRAN ERECALL subroutines are available for event recall.

CHARACTER SET C

The ASCII character set contains 128 control and graphic characters shown in the following table. Numbers, letters, and special characters that form the Cray FORTRAN character set are identified by the appearance of the letter C in the fourth column. All other characters are members of the auxiliary character set. The letter A in the fourth column of the table indicates those characters belonging to the ANSI FORTRAN character set.

The letters that appear in parentheses following the descriptions in the fifth column indicate the following control character usage.

- CC - Communication control
- FE - Format effector
- IS - Information separator

CHARACTER	ASCII OCTAL CODE	ASCII PUNCHED-CARD CODE	FORTRAN (A=ANSI) (C=CRAY)	DESCRIPTION
NUL	000	12-0-9-8-1		Null
SOH	001	12-9-1		Start of heading (CC)
STX	002	12-9-2		Start of text (CC)
ETX	003	12-9-3		End of text (CC)
EOT	004	9-7		End of transmission (CC)
ENQ	005	0-9-8-5		Enquiry (CC)
ACK	006	0-9-8-6		Acknowledge (CC)
BEL	007	0-9-8-7		Bell (audible or attention signal)
BS	010	11-9-6		Backspace (FE)
HT	011	12-9-5		Horizontal tabulation (FE)
LF	012	0-9-5		Line feed (FE)
VT	013	12-9-8-3		Vertical tabulation (FE)
FF	014	12-9-8-4		Form feed (FE)
CR	015	12-9-8-5		Carriage return (FE)
SO	016	12-9-8-6		Shift out
SI	017	12-9-8-7		Shift in
DLE	020	12-11-9-8-1		Data link escape (CC)
DC1	021	11-9-1		Device control 1
DC2	022	11-9-2		Device control 2
DC3	023	11-9-3		Device control 3
DC4	024	9-8-4		Device control 4 (stop)
NAK	025	9-8-5		Negative acknowledge (CC)
SYN	026	9-2		Synchronous idle (CC)
ETB	027	0-9-6		End of transmission block (CC)
CAN	030	11-9-8		Cancel
EM	031	11-9-8-1		End of medium
SUB	032	9-8-7		Substitute
ESC	033	0-9-7		Escape
FS	034	11-9-8-4		File separator (IS)

CHARACTER	ASCII OCTAL CODE	ASCII PUNCHED-CARD CODE	FORTRAN (A=ANSI) (C=CRAY)	DESCRIPTION
GS	035	11-9-8-5		Group separator (IS)
RS	036	11-9-8-6		Record separator (IS)
US	037	11-9-8-7		Unit separator (IS)
(Space)	040	(None)	A,C	Space (blank)
!	041	12-8-7		Exclamation mark
"	042	8-7	C	Quotation marks (diaeresis)
#	043	8-3		Number sign
$	044	11-8-3	A,C	Dollar sign (currency symbol)
%	045	0-8-4		Percent
&	046	12		Ampersand
'	047	8-5	A,C	Apostrophe (single close quotation)
(	050	12-8-5	A,C	Opening (left) parenthesis
)	051	11-8-5	A,C	Closing (right) parenthesis
*	052	11-8-4	A,C	Asterisk
+	053	12-8-6	A,C	Plus
,	054	0-8-3	A,C	Comma (cedilla)
-	055	11	A,C	Minus (hyphen)
.	056	12-8-3	A,C	Period (decimal point)
/	057	0-1	A,C	Slant (slash, virgule)
0	060	0	A,C	Zero
1	061	1	A,C	One
2	062	2	A,C	Two
3	063	3	A,C	Three
4	064	4	A,C	Four
5	065	5	A,C	Five
6	066	6	A,C	Six
7	067	7	A,C	Seven
8	070	8	A,C	Eight

CHARACTER	ASCII OCTAL CODE	ASCII PUNCHED-CARD CODE	FORTRAN (A=ANSI) (C=CRAY)	DESCRIPTION
9	071	9	A,C	Nine
:	072	8-2	A,C	Colon
;	073	11-8-6		Semicolon
<	074	12-8-4		Less than
=	075	8-6	A,C	Equal
>	076	0-8-6		Greater than
?	077	0-8-7		Question mark
@	100	8-4		Commercial at-sign
A	101	12-1	A,C	Uppercase letter
B	102	12-2	A,C	Uppercase letter
C	103	12-3	A,C	Uppercase letter
D	104	12-4	A,C	Uppercase letter
E	105	12-5	A,C	Uppercase letter
F	106	12-6	A,C	Uppercase letter
G	107	12-7	A,C	Uppercase letter
H	110	12-8	A,C	Uppercase letter
I	111	12-9	A,C	Uppercase letter
J	112	11-1	A,C	Uppercase letter
K	113	11-2	A,C	Uppercase letter
L	114	11-3	A,C	Uppercase letter
M	115	11-4	A,C	Uppercase letter
N	116	11-5	A,C	Uppercase letter
O	117	11-6	A,C	Uppercase letter
P	120	11-7	A,C	Uppercase letter
Q	121	11-8	A,C	Uppercase letter
R	122	11-9	A,C	Uppercase letter
S	123	0-2	A,C	Uppercase letter
T	124	0-3	A,C	Uppercase letter
U	125	0-4	A,C	Uppercase letter

CHARACTER	ASCII OCTAL CODE	ASCII PUNCHED-CARD CODE	FORTRAN (A=ANSI) (C=CRAY)	DESCRIPTION
V	126	0-5	A,C	Uppercase letter
W	127	0-6	A,C	Uppercase letter
X	130	0-7	A,C	Uppercase letter
Y	131	0-8	A,C	Uppercase letter
Z	132	0-9	A,C	Uppercase letter
[	133	12-8-2		Opening (left) bracket
\	134	0-8-2		Reverse slant (backslash)
]	135	11-8-2		Closing (right) bracket
^	136	11-8-7		Circumflex
_	137	0-8-5		Underline
`	140	8-1		Grave accent (single open quotation)
a	141	12-0-1	C	Lowercase letter
b	142	12-0-2	C	Lowercase letter
c	143	12-0-3	C	Lowercase letter
d	144	12-0-4	C	Lowercase letter
e	145	12-0-5	C	Lowercase letter
f	146	12-0-6	C	Lowercase letter
g	147	12-0-7	C	Lowercase letter
h	150	12-0-8	C	Lowercase letter
i	151	12-0-9	C	Lowercase letter
j	152	12-11-1	C	Lowercase letter
k	153	12-11-2	C	Lowercase letter
l	154	12-11-3	C	Lowercase letter
m	155	12-11-4	C	Lowercase letter
n	156	12-11-5	C	Lowercase letter
o	157	12-11-6	C	Lowercase letter
p	160	12-11-7	C	Lowercase letter
q	161	12-11-8	C	Lowercase letter
r	162	12-11-9	C	Lowercase letter

CHARACTER	ASCII OCTAL CODE	ASCII PUNCHED-CARD CODE	FORTRAN (A=ANSI) (C=CRAY)	DESCRIPTION
s	163	11-0-2	C	Lowercase letter
t	164	11-0-3	C	Lowercase letter
u	165	11-0-4	C	Lowercase letter
v	166	11-0-5	C	Lowercase letter
w	167	11-0-6	C	Lowercase letter
x	170	11-0-7	C	Lowercase letter
y	171	11-0-8	C	Lowercase letter
z	172	11-0-9	C	Lowercase letter
{	173	12-0		Opening (left) brace
¦	174	12-11		Vertical line
}	175	11-0		Closing (right) brace
~	176	11-0-1		Overline (tilde, general accent)
DEL	177	12-9-7		Delete

EXCHANGE PACKAGES D

An Exchange Package is a 16-word block of data in memory that is associated with a particular computer program. An Exchange Package contains the basic hardware parameters necessary to provide continuity from one execution interval for the program to the next. The CRAY-1 Exchange Package is illustrated in figure D-1; the CRAY X-MP Exchange Package is illustrated in figure C-2.

```
          0     8      16     24     32     40     48     56   63
       0  | E |   S  |R| B|          P          |         A0         |
       1  |        C      |////|     BA     |///M|  IMM    A1         |
       2  |////////////|RH|////|     LA     | M |          A2         |
       3  |///////////////|  XA  |  VL  |  F   |          A3         |
     4-7  |/////////////////////////////////////|      A4 to A7      |
    8-15  |                         S0 to S7                          |
```

Figure D-1. CRAY-1 Exchange Package

Field	Word	Bits
Error type (E)	0	0-1
Syndrome bits (S)	0	2-9
Read mode (R)	0	10-11
Bank error address (B)	0	12-15
Program register (P)	0	16-39
Chip error address (C)	1	0-15
Base address (BA)	1	18-35
Interrupt Monitor Mode bit (IMM)	1	39
High-order bits of memory error read address (RH)	2	14-15
Limit address (LA)	2	18-35
Mode bits (M)	2	36-39
Exchange address (XA)	3	16-23
Vector length (VL)	3	24-30
Flag register (F)	3	31-39
Current contents of the eight A registers	0-7	40-63
Current contents of the eight S registers	8-15	0-63

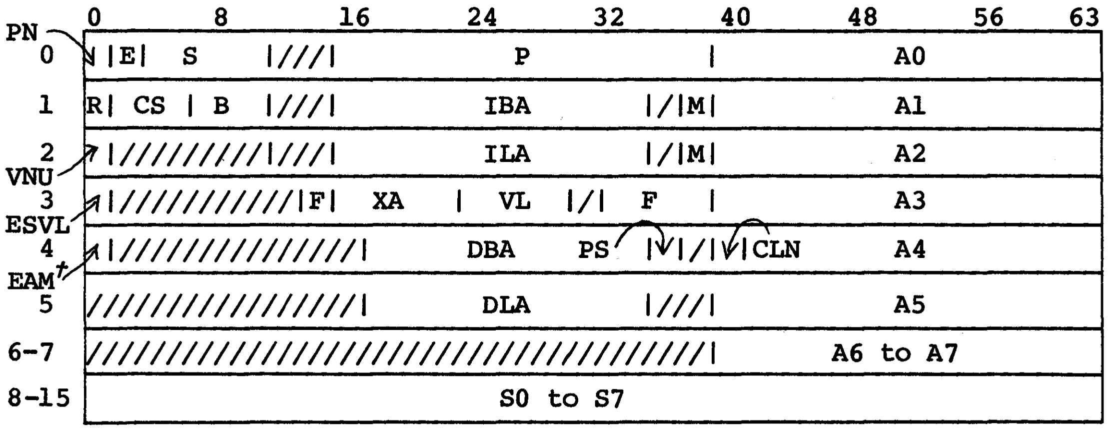

Figure D-2. CRAY X-MP Exchange Package

Field	Word	Bits: 4-Processor CRAY X-MP	Bits: Single-Processor CRAY X-MP	Bits: Dual-Processor CRAY X-MP
Processor number (PN)	0	0-1	1	1
Error type (E)	0	2-3	2-3	2-3
Syndrome bits (S)	0	4-11	4-11	4-11
Program Address register (P)	0	16-39	16-39	16-39
Read mode (R)	1	0-1	0-1	0-1
Read address (CSB)	1	2-5 (CS); 6-11 (B)	2-4 (CS); 7-11 (B)	2-6 (CS) 7-11 (B)
Instruction Base Address (IBA)	1	16-33	18-34	18-34
Instruction Limit Address (ILA)	2	16-33	18-34	18-34
Mode register (M)	1	35-39	35-37, 39	35-39
	2	35-39	35-39	35-39
Vector not used (VNU)	2	0	0	0
Enable Second Vector Logical (ESVL)†	3	0	0	0
Flag register (F)	3	14-15; 31-39	15; 31-39	14-15; 31-39
Exchange Address register (XA)	3	16-23	16-23	16-23
Vector Length register (VL)	3	24-30	24-30	24-30

† Not available on all CRAY X-MPs

Field	Word	Bits		
		4-Processor X-MP	Single-Processor X-MP	Dual-Processor X-MP
Enhanced Addressing Mode (EAM)†	4	0	NA	NA
Data Base Address (DBA)	4	16-33	18-34	18-34
Program State (PS)	4	35	35	35
Cluster Number (CLN)	4	37-39	38-39	38-39
Data Limit Address (DLA)	5	16-33	18-34	18-34
Eight A register contents	0-7	40-63	40-63	40-63
Eight S register contents	8-15	0-63	0-63	0-63

† 4-processor CRAY X-MP only

ERROR AND STATUS CODES E

SYSTEM ERROR CODES

Table E-1 describes the system error codes as released. Installation differences can change data in this table. Consult the on-site analyst for details. The CRAY-OS Message Manual, publication SR-0039, also contains additional descriptions of the abort codes and their corresponding messages.

Table E-1. Error codes for reprieve processing

System Error Code	Fatal/ Non-fatal	Reprieve Error Class (Octal Mask Value)	Description
AB001	NF	4	End-of-file on read
AB002	NF	4	Invalid LOCK or UNLOCK indicator
AB003	F	4	Device Allocation Table exhausted
AB004	NF	4	Dataset not open
AB005	NF	4	Invalid dataset open request
AB006	NF	4	No read permission
AB007	NF	4	No write permission
AB008	NF	4	Illegal bit set in RFL request word
AB009	NF	4	Attempt to delete memory outside program area
AB010	F	400	No available disk space
AB011	F	4000	System directory is full
AB012	NF	4	Job Table Area (JTA) overflow

Table E-1. Error codes for reprieve processing (continued)

System Error Code	Fatal/ Non-fatal	Reprieve Error Class (Octal Mask Value)	Description
AB013	NF	4	More memory requested than available
AB014	NF	4	More memory requested than allowed
AB015	NF	2000	Unknown acquire error
AB016	NF	2000	Subdataset $IN cannot be disposed
AB017	NF	4	Invalid dataset close request
AB018	NF	4	Dataset already opened
AB019	NOT REPRIEVABLE		Job Communication Block destroyed
AB020	NF	4	Invalid system request parameter
AB021	NF	4	Dataset not found
AB022	NF	4	Invalid program load dataset
AB023	F	200	Job time limit exceeded
AB024	F	10	Operator dropped user job
AB025	NF	2	User program requested abort
AB026	NF	4	Invalid (undefined) user request
AB027	NF	4	Call not between user BA and LA
AB028†	NF		XP errors (no message)
AB029	NF	4	Logical device name not found
AB030	NF	4	Block number error
AB031	NF	4	Unrecoverable data error

† The AB028 error code is set during abort processing when any Exchange Package error flag is set. It does not represent a single reprievable condition. One of the Exchange Package error codes (AB053 through AB058) will be set later to indicate the appropriate error.

Table E-1. Error codes for reprieve processing (continued)

System Error Code	Fatal/ Non-fatal	Reprieve Error Class (Octal Mask Value)	Description
AB032	NF	4	Unrecoverable hardware error
AB033	NF	4	Read after write or after EOD
AB034	NF	4	Unknown error
AB035	NF	4	Invalid processing direction
AB036	NF	4	Dataset prematurely terminated
AB037	NF	4	Dataset Parameter Table invalid
AB038	NOT REPRIEVABLE		Operator killed user job
AB039	NF	20	Operator reran the job
AB040	NF	4	Invalid disposition code
AB043	F	400	Allowable user log size exceeded
AB044	NF	4	Invalid dataset name
AB045	NF	400	Specified LM is too big
AB046	NF	400	Dataset size limit exceeded
AB047	NF	2000	Dataset not available from station
AB048	NF	2000	Dataset cannot be saved on a front end
AB049	NF	4	Invalid LFTs in user area
AB051	F	4	Invalid pointer to first JTA LFT
AB052	NF	4	No user LFT DN matches JTA LFT
AB053	NF	100	Floating-point error
AB054	NF	4	Operand range error
AB055	NF	4	Program range error

Table E-1. Error codes for reprieve processing (continued)

System Error Code	Fatal/ Non-fatal	Reprieve Error Class (Octal Mask Value)	Description
AB056	NF	40	Uncorrected memory error
AB057	NOT REPRIEVABLE		Interactive ABORT
AB058	F	4	Error exit
AB061	NF	4	No invoke request provided
AB062	NF	4	Invoke request abort pending
AB063	NF	4	Invoke length not multiple of 512
AB064	NF	4	Invoke length greater than maximum
AB066	NF	4	Dataset has related disposes active
AB067	NF	4	Invalid procedure dataset
AB068	NF	4	Procedure nest level exceeded
AB070	NF	10000	ATTENTION request command was entered at an interactive terminal
AB071	NF	4	Bad class structure
AB072	NF	4	DSP destroyed by user
AB073	NF	4	Undefined function code in F$INS
AB074	NF	4000	DUMPJOB processing has been inhibited
AB075	NF	4000	No permissions granted while dataset is execute-only
AB076	NF	4	Dataset is already accessed by the job
AB077	NOT REPRIEVABLE		CSP internal error
AB078	NF	4000	Privileged system request
AB079	NF	4	Unassigned JCL symbol

Table E-1. Error codes for reprieve processing (continued)

System Error Code	Fatal/ Non-fatal	Reprieve Error Class (Octal Mask Value)	Description
AB080	NF	4	Receive buffer too small
AB081	NF	4	Undefined JCL symbol
AB082	NF	4	JCL symbol cannot be modified
AB083	NF	4	Invalid message class
AB100	NF	4	Nonsequential write for tape dataset
AB101	NF	4	Interchange and unblocked are mutually exclusive
AB102	NF	4	Tape dataset can not be disposed
AB103	NF	4	VOL parameter must be equated for OLD dataset
AB104	NF	4	Job has requested more devices than it has allocated
AB105	NF	4	The Label Definition Table has been omitted for a labeled tape dataset
AB106	NF	4	The LDT has a bad field
AB107	NF	4	Unable to write trailer label group
AB108	NF	4	Write attempted on protected volume
AB109	NF	4	Attempt to write tape block larger than MBS
AB110	NF	4	Write protocol error
AB111	NF	4	Tape went off the end of the reel
AB112	NF	4	Volume is security protected
AB113	NF	4	Dataset is security protected

Table E-1. Error codes for reprieve processing (continued)

System Error Code	Fatal/ Non-fatal	Reprieve Error Class (Octal Mask Value)	Description
AB114	NF	4	Read attempted of NEW dataset
AB115	NF	4	Dataset not on this volume
AB116	NF	4	File section does not exist
AB117	NF	4	Ill-formed LDT
AB118	NF	4	Label group corrupted
AB119	NF	4	Deferred tape feature
AB120	NF	4	No HDR1 in label group
AB121	NF	4	Bad record format
AB122	NF	4	Bad blocking attributes
AB123	NF	4	Bad record length
AB124	NF	4	Bad block Length
AB125	NF	4	Bad buffer offset
AB126	NF	4	Bad owner ID
AB127	NF	4	Incomplete VSN list
AB128	NF	4	Read of expired dataset
AB129	NF	4	Write of nonexpired dataset
AB130	NF	4	Invalid expiration date
AB131	NF	4	Difference in volume block counts
AB132	NF	4	Label type can not be scratched
AB133	NF	4	F$POS is illegal for mass storage datasets
AB134	NF	4	Large block read

Table E-1. Error codes for reprieve processing (continued)

System Error Code	Fatal/ Non-fatal	Reprieve Error Class (Octal Mask Value)	Description
AB135	NF	4	Resource not available
AB136	NF	4	Tape volume/dataset access denied by servicing front-end
AB137	NF	4	Tape volume/dataset access denied due to the lack of security checks servicing front end
AB138	NF	4	Tape dataset has already been cataloged.
AB139	NF	4	Tape dataset does not reside in servicing front-end catalog.
AB140	NF	4	Update of dataset/volume state to servicing front end failed
AB141	NF	4	The tape device has been closed to user I/O.
AB142	NF	4	Tape volume does not reside in servicing front-ends catalog.
AB143	NF	4	Tape volume mount canceled by operator
AB144	NF	4	Maximum block size exceeded on write of tape dataset
AB173	NF	4	Interjob connections were open at the time of a job step advance
AB179	NF	4	DSP IN/OUT pointer not on a block boundary for WRITE/READ
AB180	NF	4	DSP buffer pointers overlap LFT/DSP area
AB181	NF	4	DSP LIMIT pointer less than FIRST
AB182	NF	4	DSP in pointer not between FIRST and LIMIT

Table E-1. Error codes for reprieve processing (continued)

System Error Code	Fatal/ Non-fatal	Reprieve Error Class (Octal Mask Value)	Description
AB183	NF	4	DSP out pointer not between FIRST and LIMIT
AB184	NF	4	DSP FIRST pointer out of bounds
AB185	NF	4	DSP LIMIT pointer out of bounds
AB186	NF	4	DSP RCW pointer not between FIRST and LIMIT
AB187	NF	4	Buffered I/O record address out of bounds
AB188	NF	4	Unknown buffer I/O function
AB189	NF	4	Buffer length not multiple of D'152
AB190	NF	4	Uncleared error in DSP
AB191	NF	4	Attempt to start I/O on busy dataset
AB192	NF	4	DSP save words destroyed by user
AB193	NF	4	All user tasks deactivated
AB194	NF	4	User deadlock detected
AB195	NF	4	Attempt to deactivate an inactive task
AB196	NF	4	Attempt to activate an active task
AB197	NF	4	User task attempt to activate itself
AB198	NF	4	Invalid user task ID
AB199	NF	4	Maximum user tasks per job exceeded

PERMANENT DATASET STATUS CODES

The permanent dataset status octal codes are placed in the PMST field of the Permanent Dataset Definition Table (PDD) which is presented in Appendix A. PMST can also be tested as the JCL symbol PDMST (see table 16-1). The PDD statuses are listed in table E-2. The logfile contains a corresponding code (of the form PD*nnn*, where *nnn* is listed in table E-2) and message for most of the status conditions.

Table E-2. PDD status

Logfile Code	PMST	Status
	1	Complete; no error.
1	11	No DNT found for the specified dataset
2	21	Maintenance permission not granted
3	31	Edition already exists
4	41	DSC full
5	51	Function code out of range
6	61	The local dataset name (DN) specified is already in use by the job
7	71	No permission granted
	101	Delay and try again
9	111	Requested dataset not in DSC
10	121	Edition does not exist
11	131	Active PDS full
12	141	Dataset not permanent
13	151	Unused
14	161	Continuation error
15	171	DAT full

Table E-2. PDD status (continued)

Logfile Code	PMST	Status
16	201	DNT full
	211	End of DSC
18	221	Specified permanent dataset already accessed by this job
	231	Request to read zero pages
	241	Invalid page number requested
21	251	No data has been written to disk
	261	SDT does not exist
	271	SDT entry not on input or output queue
	301	Unable to queue SDT entry
25	311	Dataset name in PDD is 0
26	321	Access control word validation error
27	331	Notes length exceeds allowable maximum
28	341	Unique access is not acceptable because the dataset is part of the System Directory.
29	351	Text length is zero.
30	361	The text length specified exceeds the allowable maximum.
31	371	The device on which all or part of the dataset resides is down.
	401	Error occurred while rewriting the SDT, or the SDT name and dataset type in the DSC do not match those in the PDD.
	411	Permanent dataset to be pseudo accessed is not available or the DAT in the DSC does not match the JTA DAT.

Table E-2. PDD status (continued)

Logfile Code	PMST	Status
34	421	Access is denied because crossed allocation unit exists.
35	431	The dataset is already permanent.
36	441	The DSC entry was flagged by Startup as containing a fatal error; access is denied.
	451	The DSC or DXT page buffer supplied is outside the user field length.
	461	No available QDT entries exist
	471	The dataset has outstanding disposes; do not deallocate disk space.
40	501	Allocation of multitype dataset inconsistent with related datasets
41	511	Multitype dataset has nonexistent QDT entry.
42	521	Maximum edition reached
43	531	Dataset is on an active SDT queue
44	541	Bad SDT address on Enqueue SDT request
45	551	Dataset is on a scratch device
46	561	Access denied due to DXT error
47	571	Notes length is zero.
48	601	Unused
49	611	Maximum number of DXT entries per dataset reached
50	621	Attributes dataset not local
51	631	Attributes dataset not permanent
52	641	Invalid notes buffer specified

Table E-2. PDD status (continued)

Logfile Code	PMST	Status
53	651	Invalid text buffer specified
54	661	Specified permit entry not found
	671	Invalid DXT buffer address (get/link DXT)
	701	Bad DXT linkage pointer (get/link DXT)
	711	PMPDN and DCPDN do not match (get/link DXT)
	721	Unused
	731	PMSIZE greater than maximum PDD size
	2001	Parameter error (internal to $SYSLIB)
	2002-2777	This range of status codes is reserved for magnetic tape support

GLOSSARY

GLOSSARY

A

Abort - To terminate a program or job when a condition (hardware or software) exists from which the program or computer cannot recover.

Absolute address - (1) An address permanently assigned by the machine designator to a storage location. (2) A pattern of characters that identifies a unique storage location without further modification. Synonymous with machine address.

Absolute block - Loader tables consisting of the image of a program in memory. The program image can be saved on a dataset for subsequent reloading and execution.

Address - (1) An identification, as represented by a name, label, or number, for a register, location in storage, or any other data source or destination such as the location of a station in a communication network. (2) Any part of an instruction that specifies the location of an operand for the instruction.

Allocate - To reserve an amount of some resource in a computing system for a specific purpose (usually refers to a data storage medium).

Alphabetic - A character set including, $, %, @, as well as the 26 uppercase letters A through Z.

Alphanumeric - A character set including all alphabetic characters and the digits 0 through 9.

Arithmetic operator - Part of an expression that indicates action to be performed during evaluation of expression; can be symbolic character representing addition, unary plus, subtraction, unary minus, multiplication, or division.

Assemble - To prepare an object language program from a symbolic language program by substituting machine operation codes for symbolic operation codes and absolute or relocatable addresses for symbolic instructions.

B

Base address - The starting absolute address of the memory field length assigned to the user's job. This address is maintained in the Base Address (BA) register. The base address must be a multiple of 20_8.

$BLD - A dataset on which load modules are placed by a compiler or assembler unless the user designates some other dataset.

Blank common block - A common block where data cannot be stored at load time. The first declaration need not be the largest. The blank common block is allocated after all other blocks have been processed.

Block - (1) A tape block is a collection of characters written or read as a unit. Blocks are separated by an interblock gap and can be from 1 through 1,048,576 bytes. A tape block and a physical record are synonymous on magnetic tape. (2) In COS blocked format, a block is a fixed number of contiguous characters with a block control word as the first word of the block. The internal block size for the Cray mainframe is 512 words (one sector on disk). In COS manuals, the terms tape block and 512-word block are consistently used to distinguish between the two uses.

Block control word - A word occurring at the beginning of each block in the COS blocked format that identifies the sequential position of the block in the dataset and points forward to the next block control word.

BOT - Beginning-of-tape; the position of the beginning-of-tape reflective marker.

BOV - Beginning-of-volume. See BOT.

BPI - Bits per inch. COS supports the 1600 and 6250 bpi recording densities.

Buffer - A storage device used to compensate for the difference in rate of flow of data, or time of occurrence of events, when transmitting data from one device to another. It is normally a block of memory used by the system to transmit data from one place to another. Buffers are usually associated with the I/O subsystem.

Buffer Memory - A 64-bit memory in the I/O Subsystem common to all I/O Processors.

C

Call - The transfer of control to a specified routine. The called routine normally transfers control back to the caller after the called routine has finished its task.

Card image - A one-to-one representation of the contents of a punched card, for example, a matrix where a 1 represents a punch and a 0 represents the absence of a punch. In COS blocked format, each card image is a record.

Catalog (noun) - A list or table of items with descriptive data, usually arranged so that a specific kind of information can be readily located.

Channel - A path along which signals can be sent.

Character - A logical unit composed of bits representing alphabetic, numeric, and special symbols. The Cray software processes 8-bit characters in the ASCII character set.

Code - (1) A system of character and rules representing information in a form understandable by a computer. (2) Translation of a problem into a computer language.

Common block - A block that can be declared by more than one program module during a load operation. More than one program module can specify data for a common block but if a conflict occurs, information from later programs is loaded over previously loaded information. A program can declare no common blocks or as many as 125 common blocks. The two types of common blocks are labeled and blank.

Conditional control statement block - Defines the conditions under which a group of control statements are to be processed. The statements which define the block and conditions are: IF, ELSE, ELSEIF, ENDIF, and EXITIF.

Control statement - The format, consisting of a verb and its parameters, used to control the operating system and access its products. Directives are used to control products.

Control statement input file - A dataset containing valid control statements as its first file.

Controlled device - One of one or more devices or resources which are allocated to jobs on the basis of resource limits and requests.

COS - The Cray Operating System described in this manual.

$CS - A primary control statement input file.

CSP - The Control Statement Processor (CSP) is a system program that executes in the user field. CSP initiates the job, analyzes, and stores the various elements of the control statements (that is, cracks them), processes system verbs, advances the job step by step, processes errors, and ends the job.

D

Data - (1) Information manipulated by or produced by a computer program. (2) Empirical numerical values and numerical constants used in arithmetic calculation. Data is considered to be that which is transformed by a process to produce the evidence of work. Parameters, device input, and working storage are considered data.

Dataset - A quantity of information maintained on mass storage by the Cray Operating System. Each dataset is identified by a symbolic name called a dataset name. Datasets are of two types: temporary and permanent. A temporary dataset is available only to the job that created it. A permanent dataset is available to the system and to other jobs and is maintained across system deadstarts.

Dataset characteristic information - The information that describes where the dataset resides, how large it is, its permanent name, edition number, information about the creating job, etc.

Dataset name verb - A verb that is the name of a dataset. See local or system dataset name verb.

Deadstart - The process by which an inactive machine is brought up to an operational condition ready to process jobs.

Debug - To detect, locate, and remove mistakes from a routine or malfunction of a computer. Synonymous with troubleshoot.

DEC - Disk Error Correction, a task within the STP portion of COS. DEC can be called by the Disk Queue Manager (DQM) to attempt correction of a disk error.

Delimiter - A character that separates items in a control statement or a directive; synonymous with separator.

Density - See tape density.

Device - A piece of equipment that mechanically contains and drives a recording medium.

Directive - A command used to control a product, such as UPDATE.

Diagnostic - (1) Pertaining to the detection and isolation of a malfunction or a mistake. (2) A message printed when an assembler or compiler detects a program error.

Disposition code - A code used in I/O processing to indicate the disposition to be made of a dataset when its corresponding job is terminated or the dataset is released.

DQM - The Disk Queue Manager is a task within the STP portion of COS. DQM controls the simultaneous operation of disk storage units on CPU I/O channels or on the I/O Subsystem.

Dump - (1) To copy the contents of all or part of a storage device, usually from internal storage, at a given instant of time. (2) The process of performing (1). (3) The document resulting from (1).

E

End-of-data delimiter - Indicates the end of a dataset. In COS blocked format, this is a record control word with a 17_8 in the mode field.

End-of-file delimiter - Indicates the end of a file. (1) In COS blocked format, this is a record control word with a 16_8 in the mode field. (2) On magnetic tape, this is a tapemark.

End-of-record delimiter - Indicates the end of a record. (1) In COS blocked format, this is a record control word with a 10_8 in the mode field. (2) In an ASCII punched deck, this is indicated by the end of each card.

Entry point - A location within a block that can be referenced from program blocks that do not declare the block. Each entry point has a unique name associated with it. The loader is given a list of entry points in a loader table. A block can contain any number of entry points.

An entry point name must be 1 to 8 characters and cannot contain the characters blank, asterisk, or slash. Some language processors (for example, FORTRAN) can produce entry point names under more restricted formats due to their own requirements.

EOD - End-of-data on tape. The definition of EOD is a function of whether the tape is labeled or nonlabeled and of the type of operation being performed (input or output). When reading a labeled tape, EOD is returned to the user when an EOF1 trailer label is encountered. When reading a nonlabeled tape, EOD is returned when a tapemark is read on the last volume in the volume list for a particular dataset. When writing a labeled or nonlabeled tape, EOD processing is initiated by a write EOD, rewind, close, or release request.

EOF - End-of-file on tape, sometimes used to mean end of tape trailer group.

EOI - End-of-information; see EOD.

EOT - End-of-tape; a status, set only on a write operation indicating sensing of the end of the tape reflective marker.

EOV - End-of-volume. On output, EOV occurs when end-of-tape status is returned on a write operation. This status occurs when the EOT reflective marker is sensed by the tape device. For input of a labeled tape dataset, EOV occurs when an EOV1 trailer label is read; for input of a nonlabeled dataset, EOV is returned when a tapemark is encountered and the volume list is not exhausted.

Exchange Package - A 16-word block of data in memory which is associated with a particular computer program or memory field. It contains the basic parameters necessary to provide continuity from one execution interval for the program to the next.

EXEC - The COS System Executive (EXEC) is the control center for the operating system. It alone accesses all of memory, controls the I/O channels, and selects the next program to execute.

EXP - The User Exchange Processor (EXP or UEP) is a task within the STP portion of COS. The Exchange Processor task processes all user system action requests and user error exits. The Exchange Processor also handles certain requests from the Job Scheduler (JSH) to initiate or abort a job.

Expression (JCL parameter expression) - A series of characters grouped into operands and operators which are computed as one value during parameter evaluation; should be delimited by parentheses.

External reference - A reference in one program block to an entry point in a block not declared by that program. Throughout the loading process, externals are matched to entry points (this is also referred to as satisfying externals); that is, addresses referencing externals are supplied with the correct address.

F

File - A collection of records in a dataset. In COS blocked format, a file is terminated by a record control word with 16_8 in the mode field.

Filemark - See to tapemark.

Foreign label - A special condition that can occur during the label scan at the beginning of a tape. If a NOT CAPABLE status is returned on a BOV label scan, TQM declares the tape to be foreign labeled (FRN) which protects a 7-track tape or a 9-track, 800 bpi tape from being accidently destroyed.

Formal parameter specifications - Parameters in a procedure definition which identify the character strings within the procedure body that can be substituted during the procedure's evaluation.

Front-end dataset servicing - The act of requesting and receiving information concerning a particular dataset that is known to the front-end computer system. Typical servicing is:

- Direct operator messages concerning tape volume/drive activity,

- Obtaining required information concerning a dataset, such as what volumes it resides on, the expiration date of each volume, access permissions, etc., and

- Updating information for a dataset and/or tape volume for use by that computer system.

Front-end processor - A computer connected to a Cray Computer System channel. The front-end processor supplies data and jobs to the Cray mainframe and processes or distributes the output from the jobs. Front-end systems are also referred to as stations in Cray publications.

G

Generic resource - A device or group of devices connected to the Cray system which is accessible to user jobs. Devices which constitute a generic resource are characterized by common attributes, such as tape drives with 6250 bpi capability. These devices are subject to regulated access by the system.

H

Heap - An area of memory within the user field managed by user-callable library routines. The heap provides dynamic storage allocation for a single job.

HLM - High limit of memory, the highest relative memory address available to the user for program and data area.

I

$IN - A dataset containing the job control language statements as well as the source input and data for compilers and assemblers, unless the user designates some other dataset (FT05 for example).

In-line procedure - A procedure defined in a control statement file.

Input/Output - (1) Commonly called I/O. To communicate from external equipment to the computer and vice versa. (2) The data involved in such a communication. (3) Equipment used to communicate with a computer. (4) The media carrying the data for input/output.

Integer constant - Specifies an octal value or a decimal value that can be signed as positive or negative.

Interchange format - One of the two ways in which tape datasets can be read or written. Each tape block of data corresponds to a single logical record in COS blocked format. Interchange format is selected by setting DF=IC when a tape dataset is accessed. As far as I/O routines in the Cray mainframe are concerned, interchange datasets must be in COS blocked format because the COS blocked structure (BCWs and RCWs) is used to describe each tape block read or written. This blocked structure allows the user to write or read variable-length tape blocks at high speed with data resolution to the 8-bit byte level of the tape device. The record control word (RCW) is used to define the tape block length on output and to describe the block length on input. No BCW or RCW ever appears in the data written on the tape.

Interblock gaps - The physical separation between successive tape blocks on magnetic tape.

I/O Subsystem - Part of a CRAY-1 S Series Model S/1200 through S/4400, all models of the CRAY-1 M Series and CRAY X-MP Computer Systems consisting of two to four I/O processors and one-half, one, four, or eight million words of shared Buffer Memory. The optional tape subsystem is composed of at least one block multiplexer channel, one tape controller, and two tape units. The tape units supported are IBM-compatible 9-track, 200 ips, 1600/6250 bpi devices.

Iterative control statement block - Defines the repeated execution of a series of statements if a condition is satisfied.

J

JCL block control statement - A statement in the control statement file that is part of a group of control statements called a block which specifies an action to be taken by COS; the three types of blocks are: procedure defintion, conditional, and iterative.

JCM - The Job Class Monitor is a task within the STP portion of COS. JCM assigns every job to a job class (see JOB statement description) before it enters the input queue.

Job - (1) An arbitrarily defined parcel of work submitted to a computing system. (2) A collection of tasks submitted to the system and treated by the system as an entity. A job is presented to the system as a formatted dataset. With respect to a job, the system is parametrically controlled by the content of the job dataset.

Job Communication Block - The first 200_8 words of the job memory field. This area is used to hold the current control statement and certain job-related parameters. The area is accessible to the user, the operating system, and the loader for inter-phase job communication.

Job control statement - Any of the statements used to direct the operating system in its functioning, as compared to data, programs, or other information needed to process a job but not intended directly for the operating system itself. A control statement can be expressed in card, card image, or user terminal keyboard entry medium.

Job deck - The physical representation of a job before processing either as a deck of cards or as a group of records. The first file of the job dataset contains the job statements and the job parameters which will be used to control the job. Following files contain the program and data which the job will require for the various job control statements. The job deck is terminated by an end-of-data delimiter.

Job input dataset - A dataset named $IN on which the card images of the job deck are maintained. This consists of programs and data referenced by various job steps. The user can manipulate the dataset like any other dataset (excluding write operations).

Job output dataset - Any of a set of datasets recognized by the system by a special dataset name (for example, $OUT, $PLOT, and $PUNCH), which becomes a system permanent dataset at job end and is automatically staged to a front-end computer for processing.

Job step - A unit of work within a job, such as source language compilation or object program execution.

JSH - The Job Scheduler (JSH) is a task within the STP portion of COS. The Job Scheduler task initiates the processing of a job, selects the currently active job, manages job roll-in and roll-out, and terminates a job.

K

Keyword parameter - A string of 1 to 8 alphanumeric characters that consists of a keyword followed by one or more values; identified by its form rather than by its position in the control statement.

L

$LOG - See logfile.

Label group - A group of tables that precede and follow the user data at dataset and/or volume boundary conditions. The label group describes the characteristics of the volume or dataset.

Labeled common - A common block into which data can be stored at load time.

Library - A dataset composed of sequentially organized records and files. The last file of the library contains a library directory. The rest of the files and records, known as entries, can consist of processed procedure definitions and/or relocatable modules. The directory gives a listing of entry names with their associated characteristics.

Library-defined verb - A 1- through 8-character name of a program or procedure definition residing in a library that is a part of the current library searchlist.

Limit address - The upper address of a memory field. This address is maintained in the limit address (LA) register.

Literal - A symbol which names, describes, or defines itself and not something else that it might represent.

Literal constant - A string of 1 through 8 characters delimited with apostrophes whose ordinal numbers are in the range 040_8 through 176_8; value of a character constant corresponds to the ASCII character codes positioned within a 64-bit word; alignment indicated can be left- or right-adjusted and zero-filled or left-adjusted and space-filled; apostrophes remain as part of value.

Literal string - A string delimited with apostrophes which are normally not treated as part of the value, except with JCL block control statements which treat the apostrophes as part of the string value.

Loader tables - The form in which code is presented to the loader. Loader tables are generated by compilers and assemblers according to loader requirements. The tables contain information required for loading such as type of code, names, types and lengths of storage blocks, data to be stored, etc.

Loading - The placement of instructions and data into memory so that it is ready for execution. Loader input is obtained from one or more datasets and/or libraries. Upon completion of loading, execution of the program in the job's memory field is optionally initiated. Loading can also involve the performance of load-related services such as generation of a loader map, presetting of unused memory to a user-specified value, and generation of overlays.

Load point - See BOT.

Local dataset - A temporary or permanent dataset accessible by the user.

Local dataset name verb - A verb that is the name of a local dataset consisting of an alphabetic character followed by 1 through 6 alphanumeric characters. Requests that COS load and execute an absolute binary program from the first record of the named dataset.

Logfile - During the processing of the job, a special dataset named $LOG is maintained. At job termination, this dataset is appended to the $OUT file for the job. The job logfile serves as a time-ordered record of the activities of the job: all control statements processed by the job, significant information such as dataset usage, all operator interactions with a job, and errors detected during processing of the job.

Logical operator - Represents logical function performed on operands on a bit-by-bit basis, returning a 64-bit result; functions are: inclusive OR, intersection, exclusive OR, unary complement.

M

Macro instruction - An instruction in a source language that is equivalent to a specified sequence of machine instructions.

Magnetic tape - A tape with a magnetic surface on which data can be stored by selective polarization of portions of that surface.

Mainframe - The central processor of the computer system. It contains the arithmetic unit and special register groups. It does not include input, output, or peripheral units and usually does not include internal storage. Synonymous with central processing unit (CPU).

Mass storage - The storage of a large amount of data that is also readily accessible to the central processing unit of a computer.

MEP - The Error Message Processor (MEP) is a task within the STP portion of COS. Error messages are passed from the System Executive (EXEC) to the Log Manager (MSG) through the Error Message Processor.

MSG - The Log Manager (MSG) is a task within the STP portion of COS. MSG writes messages in the system and user logfiles.

Multiprocessing - Use of several computers to logically or functionally divide jobs or processes; and to execute various programs or segments asynchronously and simultaneously.

Multiprogramming - A technique for handling multiple routines or programs simultaneously by overlapping or interleaving their execution, that is, permitting more than one program to time-share machine components.

Multitasking - A type of multiprocessing in which more than one task may be simultaneously active for a single job.

N

Nesting - Including a block of statements of one kind into a larger block of statements of the same kind, such as an iterative block within a larger iterative block.

Not Capable - A tape status indicating the reel currently mounted cannot be read by the control unit and drive. The Not Capable status would be returned if an 800 bpi tape were mounted on a device that supported only 1600 and 6250 bpi, for example. Since it is not possible to read a Not Capable tape to verify label type and contents, COS rejects (unloads) all tapes that return a Not Capable status.

O

$OUT - A dataset that contains the list output from compilers and assemblers unless the user designates some other dataset. At job end, the job logfile is added to the $OUT dataset and the dataset is sent to a front-end computer.

Operand - A character string in an expression that is operated on during evaluation; types are integer constant, literal constant, symbolic variable, and subexpresion.

Operating system - (1) The executive, monitor, utility, and any other routines necessary for the performance of a computer system. (2) A resident executive program that automates certain aspects of machine operation, particularly as they relate to initiating and controlling the processing of jobs.

Operator - A symbolic representation indicating the action to be performed in an expression; types are arithmetic, relational, and logical operators.

Overlaying - A technique for bringing routines into memory from some other form of storage during processing so that several routines will occupy the same storage locations at different times. Overlaying is used when the total memory requirements for instructions exceeds the available memory.

OVM - The Overlay Manager (OVM) is a part of the STP portion of COS and manages the use of the overlaid portion of COS itself.

P

$PROC - A dataset to which in-line procedure definitions are written.

Parallel processing - Simultaneous or approximately simultaneous processing of jobs, job steps, programs, and parts of programs.

Parameter - A quantity in a control statement which can be given different values when the control statement is used for a specific purpose or process.

Parcel - A 16-bit portion of a word which is addressable for instruction execution but not for operand references. An instruction occupies one or two parcels; if it occupies two parcels, they can be in separate words.

Parenthetic string - A string delimited with parentheses instead of apostrophes; parentheses are treated as part of the string when evaluated except when preceded by an initial, parameter, equivalence, or concatenation separator character.

PDM - The Permanent Dataset Manager (PDM) is a task within the STP portion of COS and provides the means for creating, accessing, deleting, maintaining, and auditing disk-resident permanent datasets.

Permanent dataset - A dataset known to the operating system as being permanent; the dataset survives deadstart.

Positional parameter - A parameter that must appear in a precise position relative to the separators in the control statement.

Procedure - A named sequence of control statements and/or data that is saved in a library for processing at a later time when activated by a call to its name by a calling statement; provides the capability of replacing values within the procedure with other values.

Procedure definition - The definition of a procedure saved in a library to be called for processing at a later time; if defined in a job control statement is called an in-line procedure definition.

Program - (1) A sequence of coded instructions that solves a problem. (2) To plan the procedures for solving a problem. This can involve analyzing the problem, preparing a flow diagram, providing details, developing and testing subroutines, allocating storage, specifying I/O formats, and incorporating a computer run into a complete data processing system.

Program block - The block within a load module usually containing executable code. It is automatically declared for each program (though it can be zero-length). It is local to the module; that is, it can be accessed from other load modules only through use of external symbols. Data placed in a program block always comes from its own load module.

Program name - Also referred to as IDENT name or deck name, the name contained in the loader PDT table at the beginning of each load module.

Program library - (PL) The base dataset used by the UPDATE utility. This dataset consists of one or more specially formatted card image *decks*, each separated by an end-of-file.

R

Record - A group of contiguous words or characters related to each other by virtue of convention. A record is fixed or variable length. (1) In COS blocked format, a record ends with a record control word with 10_8 in the mode field. (2) In an ASCII-coded punched deck, each card is a record. (3) For a listable dataset, each line is a record. (4) For a binary load dataset, each module is a record.

Relational operator - An operator that indicates the comparison to be performed between the operands in an expression (-1 for a TRUE result and 0 for a FALSE result); types are equal, not equal, less than, greater than, less than or equal, and greater than or equal.

Relative address - An address defined by its relationship to a base address (BA) such that the base address has a relative address of 0.

Relocatable address - An address presented to the loader in such a form that it can be loaded anywhere in the memory field. A relocatable address is defined as being relative to the beginning address of a load module program block or common block.

Relocatable module - This is the basic program unit produced by a compiler or assembler. CAL produces a relocatable module from source statements delineated by IDENT and END. In FORTRAN, the corresponding beginning statements are PROGRAM, SUBROUTINE, BLOCK DATA, or FUNCTION. The corresponding end statement is END.

A relocatable module consists of several loader tables that define blocks, their contents, and address relocation information.

Relocate - In programming, to move a routine from one portion of internal storage to another and to adjust the necessary address references so that the routine can be executed in its new location. Instruction addresses are modified relative to a fixed point or origin. If the instruction is modified using an address below the reference point, relocation is negative. If addresses are above the reference point, relocation is positive. Generally, a program is loaded using positive relocation.

S

SCP - The Station Call Processor (SCP) is a task within the STP portion of COS and handles communications with front-end computer systems.

Sector - A physical area on disk equivalent to 512 Cray words. In COS blocked format, a block is also 512 contiguous words with a block control word as the first word of the block. Therefore the internal block size for the Cray is equivalent to one Cray disk sector. This is the unit of data transfer between the Cray mainframe and the I/O Subsystem.

SPM - The System Performance Monitor (SPM) is the task within COS that collects and reports statistics about COS system performance.

STG - Stager (STG) task is a subtask of SCP within the STP portion of COS that handles dataset transfers between the Cray mainframe and its front-end processors.

STP - The System Task Processor (STP) is the main portion of the COS operating system and consists of tables, a set of routines called tasks, and some re-entrant routines common to all tasks.

Separator - Synonym for delimiter.

String - A sequence of characters delimited by apostrophes or parentheses which is taken literally as a parameter value; see literal string and parenthetic string.

Subexpression - An expression that is evaluated so that its result becomes an operand.

Substitution parameters - Parameters on procedure definition prototype statement or procedure calling statement which provide replacement values to be substituted during evaluation for strings flagged within the procedure body.

Symbolic variable - A string of 1 to 8 alphanumeric characters, beginning with an alpha character that represents values maintained by COS and/or the user.

System dataset name verb - A verb that is the name of a system-defined dataset in the System Directory Table (SDR); consists of an alphabetic character which can be followed by 1 through 6 alphanumeric characters.

System logfile - A permanent dataset named $SYSTEMLOG.

System verb - Requests that COS perform a function; consists of an alphabetic character which can be followed by 1 through 6 alphanumeric characters

T

Table - A collection of data, each item being uniquely identified either by some label or by its relative position.

Tape block - A group of contiguous characters recorded on and read from magnetic tape as a unit.

Tape control unit - A piece of equipment connected to a block multiplexer channel that provides the capability for controlling the operation of one or more tape devices. Up to four control units can be combined to drive a maximum of 16 tape devices. The control units are cross connected to all devices. Such a configuration is called a 4x16 (four by sixteen). If one control unit were to be connected to three devices, it would be referred to as a 1x3 configuration.

Tape density (bpi) - The number of bits per inch on magnetic tape. COS supports 6250 bpi and 1600 bpi.

Tape format - The way tape datasets are read or written. In *interchange format*, each tape block of data corresponds to a single logical record in COS blocked format. In *transparent format*, each tape block is a fixed multiple of 512 words based on the density of the tape.

Tape volume - A reel of magnetic tape.

Tapemark - A special hardware bit configuration recorded on magnetic tape. It indicates the boundary between combinations of datasets and labels. It is sometimes called a filemark.

Task - A subprogram or uniquely named process that can have code and data areas in common with other tasks of the same job. A task is a unit of computation that can be scheduled independently of other tasks in the same job step. A job step can consist of a single task, or it may consist of several tasks running in parallel with each other.

Temporary dataset - A dataset which is not permanent and is available only to the job that created it.

Time slice - The maximum amount of time during which the CPU can be assigned to a job without re-evaluation as to which job should have the CPU next.

Timestamp - A 1-word binary number that represents specific date and time. Timestamps are expressed as the number of (nanosecond/1.024) units between the date/and time in question and midnight, 1 January 1973. Timestamps appear in machine-independent tables used by the operating system.

TQM - The Tape Queue Manager (TQM) is the System Task Processor (STP) task that manages tape I/O between one or more user jobs and the I/O Subsystem.

Track - The smallest amount of disk space which can be allocated or deallocated by COS. A track is equivalent to 18 sectors for DD-19, DD-29, Buffer Memory and Solid-state storage device.

Transparent format - One of two ways tape datasets are read or written. Each tape block is a fixed multiple of 512 words. Transparent format is the default tape dataset format and is designated by setting DF=TR when accessing a tape dataset. This format produces a fixed-length block dataset (16384 bytes at 1600 bpi or 32768 bytes at 6250 bpi) that can be a COS blocked or unblocked dataset as far as any I/O routines are concerned. The tape subsytem merely takes four (1600 bpi) or eight (6250 bpi) sectors and processes them as one physical tape block. When a short block is read, it is considered to be EOD.

U

UEP - User Exchange Processor. See EXP.

Unit record device - A device such as a card reader, printer, or card punch for which each unit of data to be processed is considered a record.

Unload - To remove a tape from ready status by rewinding beyond the load point. The tape is then no longer under control of the computer.

Unsatisfied external - An external reference for which the loader has not yet loaded a module containing the matching entry point.

User field - A portion of memory containing instructions and data defined for a specific job. Field limits are defined by the base address and the limit address. A program cannot execute outside of its field nor refer to operands outside of its field.

User logfile - A dataset named $LOG created for a job when it is initiated by the Job Scheduler.

V

Verb - The first nonblank field of a control statement; specifies the action to be taken by COS during control statement evaluation.

Volume - A physical unit of storage media that can be dismounted from a storage device, for example, a reel of magnetic tape.

Volume identifier - Up to 6 alphanumeric characters used to identify a physical reel of tape. On labeled tapes, the volume identifier is actually recorded on tape in the volume header label. Volume identifier is synonomous with volume serial number.

VSN - Volume serial number. See volume identifier.

W

Word - A group of bits between boundaries imposed by the computer. Word size must be considered in the implementation of logical divisions such as character. The word size of the CRAY-1 and CRAY X-MP computers is 64 bits.

SUMMARY

SUMMARY

This summary lists control statements in the COS job control language.

A parameter shown in all UPPERCASE letters must be coded literally, while a value must be substituted for an *italicized* item. Braces enclose alternate choices.

The column at the left margin refers to the location in this manual of additional information on each control statement. References in the form 1-1 indicate a section and page in this manual. Other references are to the pulication numbers of CRI manuals in which you can find the control statements described.

Reference	Control statement
7-7	* *comment text*
9-5	ACCESS,DN=*dn*,NA,ERR,MSG,IR,PDN=*pdn*,ID=*uid*,ED=*ed*,R=*rd*,W=*wt*,M=*mn*, UQ,LE,OWN=*ov*,DT=*dt*,NEW,RING=$\begin{Bmatrix}\text{IN}\\\text{OUT}\end{Bmatrix}$,DEN=*den*,MF=*fes*, VOL=vol_1:vol_2:...vol_n,=*fsec*,FSEC=*fsec*,LB=*lb*,DF=*df*,PROT,MBS=*mbs*, MOD,XDT=*yyddd*,RT=*rt*,FD=*fd*,CV=*cv*,CS=*cs*,F=*f*,RF=*rf*,RS=*rs*,FSEQ=*fseq*.
7-14	ACCOUNT,AC=*ac*,APW=*apw*,NAPW=*napw*,US=*us*,UPW=*upw*,NUPW=*nupw*.
10-1	ACQUIRE,DN=*dn*,PDN=*pdn*,ID=*uid*,ED=*ed*,RT=*rt*,R=*rd*,W=*wt*,M=*mn*,UQ, TEXT=*text*,MF=*mf*,TID=*tid*,DF=*df*,OWN=*own*,PAM=*mode*,ADN=*adn*(*m*), TA=*opt*,NOTES=*notes*,ERR,MSG.
9-15	ADJUST,DN=*dn*,NA,ERR,MSG.
SR-0036	APML,CPU=*type*,I=*idn*,L=*ldn*,B=*bdn*,E=*edn*,ABORT,DEBUG,*options*, LIST=*nn*,S=*sdn*,SYM=*sym*,T=*bst*,X=*xdm*.
8-1	ASSIGN,DN=*dn*,S=*size*,SZ=*size*,NOF,BS=*blk*,DV=*ldv*,DT=*dt*,DF=*df*,RDM,U, MR,LM=*lm*,INC=*nds*,C,DC=*dc*,BFI=*bfi*,A=*un*,FD=*fd*,CV=*cv*,CS=*cs*, F=*f*,RF=*rf*,RS=*rs*,MBS=*mbs*.
11-8	AUDIT,L=*ldn*,B=*bdn*,PDN=*pdn*,ID=*uid*,US=*usn*,DV=*dvn*,SZ=*dsz*, ACC=*opt*:*opt*,X=*mm/dd/yy*:'*hh*:*mm*:*ss*',TCR=*mm/dd/yy*:'*hh*:*mm*:*ss*', TLA=*mm/dd/yy*:'*hh*:*mm*:*ss*',TLM=*mm/dd/yy*:'*hh*:*mm*:*ss*',CW=*cw*, OWN=*ov*,LO=*opt*:*opt*:*opt*:*opt*:*opt*,BO=*opt*:*opt*:*opt*:*opt*.
SR-0013	AUDPL,P=*pdn*,I=*idn*,L=*ldn*,M=*mdn*,B=*bdn*,*=*m*,/=*c*,DW=*dw*,LW=*lw*,JU=*ju*,DK=*list*, PM=*list*,LO=*string*,CM,NA,NR.
	BIND,OAL=*oaldn*,NAL=*naldn*,L=*ldn*,DEBUG=*i*,NA.
15-1	BUILD,I=*idn*,L=*ldn*,OBL=*odn*,B=*bdn*,NBL=*ndn*,SORT,NODIR,REPLACE.
SR-0000	CAL,CPU=*type*,I=*idn*,L=*ldn*,B=*bdn*,E=*edn*,ABORT,DEBUG,*options*,LIST=*nn*, S=*sdn*,SYM=*sym*,T=*bst*,X=*xdm*.
7-10	CALL,DN=*dn*[,CNS].
SR-0033	CFT,I=*idn*,L=*ldn*,B=*bdn*,C=*cdn*,E=*n*,EDN=*edn*,ON=*string*,OFF=*string*,MAXBLOCK=*mb*, TRUNC=*nn*,AIDS=*aids*,OPT=*option*,UNROLL=*r*,INT=*il*,ALLOC=*allocation*, CPU=*cputype*:*characteristics*,DEBUG,SAVEALL,ANSI.
7-16	CHARGES,SR=*options*.
13-12	COMPARE,A=*adn*,B=*bdn*,L=*ldn*,DF=*df*,ME=*maxe*,CP=*cpn*,CS=*csn*, CW=cw_1:cw_2,ABORT=*ac*.
SG-0094	CONNECT,DN=*dn*[,DV=*dv*][,MF=*mf*][,DF=*df*][,TEXT=*text*][,STEXT=*stext*].
12-2	COPYD,I=*idn*,O=*odn*.
12-2	COPYF,COPYF,I=*idn*,O=*odn*,NF=*n*.

Reference	Control statement
12-1	COPYR,I=*idn*,O=*odn*,NR=*n*.
12-3	COPYU,I=*i*,O=*o*,NS=*ns*.
SM-0072	CSIM,I=*idn*,L=*ldn*,T=*time*,SYM=sym_1:sym_2:sym_3,MSG=*msg*.
16-28	&DATA,*dn*.
13-6	DEBUG,I=*idn*,O=*odn*,DUMP=*ddn*,TRACE=*n*,SYMS=*sym*,NOTSYMS=*nsym*, MAXDIM=*dim*,BLOCKS=*blk*,NOTBLKS=*nblk*,PAGES=*np*, COMMENTS='*string*'.
9-19	DELETE,DN=*dn*,NA,ERR,MSG,PARTIAL.
10-5	DISPOSE,DN=*dn*,SDN=*sdn*,DC=*dc*,DF=*df*,MF=*mf*,SF=*sf*,ID=*uid*,TID=*tid*, ED=*ed*,RT=*rt*,R=*rd*,W=*wt*,M=*mn*,TEXT=*text*,WAIT,NOWAIT,DEFER,NRLS.
13-9	DSDUMP,I=*idn*,O=*odn*,DF=*df*,IW=*n*,NW=*n*,IR=*n*,NR=*n*,IF=*n*,NF=*n*,IS=*n*,NS=*n*, Z,DB=*db*,DSZ=*sz*.
13-2	DUMP,I=*idn*,O=*odn*,FWA=*fwa*,LWA=*lwa*,JTA,NXP,V,DSP,FORMAT=*f*,CENTER.
13-1	DUMPJOB.
7-19	ECHO,ON=$class_1$:...:$class_n$,OFF=$class_1$:...:$class_n$.
16-3	ELSE.
16-3	ELSEIF(*expression*)
16-2	ENDIF.
16-12	ENDLOOP.
16-28	ENDPROC.
7-5	EXIT.
16-4	EXITIF.
16-4	EXITIF(*expression*)
16-12	EXITLOOP.
16-12	EXITLOOP(*expression*)
10-11	FETCH,DN=*dn*,SDN=*sdn*,TEXT=*text*,MF=*mf*,TID=*tid*,DF=*df*.
13-15	FLODUMP.
13-17	FTREF,I=*idn*,L=*idn*,TREE=*op*,CB=*op*,ROOT=*root*,END=*end*,LEVEL=*n*, DIR=*dir*,NORDER.
16-2	IF(*expression*)

Reference	Control statement
7-9	IOAREA, $\begin{Bmatrix}\text{LOCK}\\\text{UNLOCK}\end{Bmatrix}$.
SG-0094	ISP[,MF=*mf*][,TEXT=*text*][,STEXT=*stext*].
13-24	ITEMIZE,DN=*dn*,L=*odn*,NREW,NF=*n*,T,BL,E,B,X.
7-1	JOB,JN=*jn*,MFL=*fl*,T=*tl*,P=*p*,US=*us*,OLM=*olm*,CL=*jcn*,*gn*=*nr*.
14-1 SG-0056	LDR,DN=*dn*,LIB=*ldn*,NOLIB=*ldn*,LLD,AB=*adn*,MAP=*op*,SID[='*string*'], T=*tra*,NX,DEB=*l*,C=*com*,OVL=*dir*,CNS,NA,USA,L=*ldn*, SET=*val*,E=*n*,I=*sdir*,NOECHO,SECURE, GRANT=sc_1:sc_2:...:sc_n,BC=*bc*,PAD=*pad*,NORED, STK[=*initial size*[:*increment*]],MM[=*initial size*[:*increment*]], MMEPS=*epsilon*,MMLOC=$\begin{Bmatrix}\text{AFTER}\\\text{BEFORE}\end{Bmatrix}$.
7-20	LIBRARY,DN=dn_1:dn_2...:dn_{64},V.
16-12	LOOP.
7-5	MEMORY[,FL=*fl*] $\left[\begin{Bmatrix}\text{,USER}\\\text{AUTO}\end{Bmatrix}\right]$.
7-3	MODE,FI=*option*,BT=*option*,ORI=*option*,EMA=*option*,AVL=*option*.
9-16	MODIFY,DN=*dn*,PDN=*pdn*,ID=*uid*,ED=*ed*,RT=*rt*,R=*rd*,W=*wt*,M=*mn*,NA,ERR, MSG,EXO=$\begin{Bmatrix}\text{ON}\\\text{OFF}\end{Bmatrix}$,PAM=*mode*,TA=*opt*,TEXT=*text*,NOTES=*notes*.
7-7	NORERUN, $\begin{Bmatrix}\text{ENABLE}\\\text{DISABLE}\end{Bmatrix}$.
7-21	OPTION,[LPP=*n*,] STAT=$\begin{Bmatrix}\text{ON}\\\text{OFF}\end{Bmatrix}$.
SR-0060	PASCAL,I=*idn*,L=*ldn*,B=*bdn*,O=*list*.
11-2	PDSDUMP,DN=*dn*,DV=*ldv*, $\begin{Bmatrix}\text{PDN}\\\text{PDS}\end{Bmatrix}$=*pdn*,ED=*ed*,CW=*cw*,ID=*uid*,US=*usn*,OWN=*ov*, INC=*mm/dd/yy*:'*hh*:*mm*:*ss*',ARC=*mm/dd/yy*:'*hh*:*mm*:*ss*', TS=*opt*,X,C,D,I,O,S,B,SO.
11-5	PDSLOAD,L=*ldn*,DN=*dn*, $\begin{Bmatrix}\text{PDN}\\\text{PDS}\end{Bmatrix}$=*pds*,ED=*ed*,CW=*cw*,ID=*uid*,NID=*nuid*, US=*usn*,OWN=*ov*,NOWN=*nov*,DV=*dvn*,RP,CR,A,I,O,S,NA,SO,TLA.
9-20	PERMIT,PDN=*pdn*,ID=*uid*,AM=*m*,RP,USER=*ov*,ADN=*adn*,NA,ERR,MSG.
13-14	PRINT(*expression*)
16-26	PROC.
8-10	RELEASE,DN=dn_1:dn_2:...:dn_8,HOLD.

Reference	Control statement
7-8	RERUN,{ENABLE / DISABLE}.
7-13	RETURN[,ABORT].
12-6	REWIND,DN=dn_1:dn_2:...:dn_8.
7-17	ROLLJOB.
9-2	SAVE,DN=*dn*,PDN=*pdn*,ID=*uid*,ED=*ed*,RT=*rt*,R=*rd*,W=*wt*,M=*mn*,UQ,NA,ERR, MSG,EXO={ON / OFF},PAM=*mode*,ADN=*adn*(*m*),TA=*opt*,TEXT=*text*, NOTES=*notes*.
SR-0066	SEGLDR,I=*idn*,L=*ldn*,DW=*dw*,CMD='*dirstr*'.
SR-0066	SEGRLS.
7-18	SET(*symbol*=*expression*)
SG-0056	SID=*adn*,I=*idn*,L=*ldn*,ECH=*edn*,CNT=*n*.
12-5	SKIPD,DN=*dn*.
12-4	SKIPF,DN=*dn*,NF=*n*.
12-4	SKIPR,DN=*dn*,NR=*n*.
12-6	SKIPU,DN=*dn*,NS=*ns*.
SR-0033	SKOL,I=*idn*,L=*ldn*,X=*xdn*,E=*edn*,O=*odn*,M=*mdn*,VIEW,LISTOFF,''''=*x*, @=*y*,#=*z*.
10-10	SUBMIT,DN=*dn*,SID=*sf*,DID=*df*,TID=*tid*,DEFER,NRLS.
7-7	SWITCH,*n*=*x*.
13-22	SYSREF,X=*xdn*,L=*ldn*.
SG-0055	TEDI,DN=*dn*,I=*idn*,L=*ldn*.
SR-0013	UPDATE,P=*pdn*,I=idn_1:idn_2:...:idn_n,C=*cdn*,N=*ndn*,L=*ldn*,E=*edn*, S=*sdn*,*=*m*,/=*c*,DW=*dw*,DC=*dc*,FML=*n*,Q[=d_1:d_2:...:d_n, options. Q='d_1,d_2,...,d_j.d_k,...,d_n'
12-7	WRITEDS,DN=*dn*,NR=*nr*,RL=*rl*.

INDEX

INDEX

READERS COMMENT FORM

CRAY-OS Version 1 Reference Manual SR-0011 M

Your comments help us to improve the quality and usefulness of our publications. Please use the space provided below to share with us your comments. When possible, please give specific page and paragraph references.

NAME ______________________________

JOB TITLE ______________________________

FIRM ______________________________

ADDRESS ______________________________

CITY ______________ STATE ________ ZIP ________

CRAY RESEARCH, INC.

CUT ALONG THIS LINE

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD
FIRST CLASS PERMIT NO 6184 ST. PAUL, MN

POSTAGE WILL BE PAID BY ADDRESSEE

CRAY RESEARCH, INC.

1440 Northland Drive
Mendota Heights, MN 55120
U.S.A.

Attention:
PUBLICATIONS

STAPLE

READERS COMMENT FORM

CRAY-OS Version 1 Reference Manual SR-0011 M

Your comments help us to improve the quality and usefulness of our publications. Please use the space provided below to share with us your comments. When possible, please give specific page and paragraph references.

NAME ____________________

JOB TITLE ____________________

FIRM ____________________

ADDRESS ____________________

CITY ____________ STATE ________ ZIP ________

CRAY RESEARCH, INC.

CUT ALONG THIS LINE

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD

FIRST CLASS PERMIT NO 6184 ST. PAUL, MN

POSTAGE WILL BE PAID BY ADDRESSEE

CRAY RESEARCH, INC.

1440 Northland Drive
Mendota Heights, MN 55120
U.S.A.

Attention:
PUBLICATIONS

STAPLE