DNOS

Supervisor Call (SVC) Reference Manual

Part No. 2270507-9701 *C
March 1985

TEXAS INSTRUMENTS

LIST OF EFFECTIVE PAGES

INSERT LATEST CHANGED PAGES AND DISCARD SUPERSEDED PAGES

Note: The changes in the text are indicated by a change number at the bottom of the page and a vertical bar in the outer margin of the changed page. A change number at the bottom of the page but no change bar indicates either a deletion or a page layout change.

DNOS Supervisor Call (SVC) Reference Manual (2270507-9701)

Original Issue	August 1981
Revision	October 1982
Revision	November 1983
Change 1	March 1985

The total number of pages in this publication is 670 consisting of the following:

PAGE NO.	CHANGE NO.	PAGE NO.	CHANGE NO.	PAGE NO.	CHANGE NO.
Cover	1	6-4	1	7-41 - 7-42	0
Effective Pages	1	6-4A/6-4B	1	7-43	1
Eff. Pages Cont.	1	6-5 - 6-18	0	7-44 - 7-67	0
iii - iv	1	6-19	1	7-68	1
v - xviii	0	6-20 - 6-25	0	7-69 - 7-93	0
1-1 - 1-8	0	6-26	1	7-94	1
2-1 - 2-14	0	6-27 - 6-138	0	7-95 - 7-138	0
3-1 - 3-20	0	6-139	1	8-1 - 8-6	0
4-1 - 4-9	0	6-140 - 6-163	0	8-7	1
4-10	1	6-164	1	8-8 - 8-21	0
4-11 - 4-14	0	6-165 - 6-178	0	8-22	1
5-1	0	7-1 - 7-2	0	8-23 - 8-30	0
5-2	1	7-3 - 7-4	1	8-31	1
5-3 - 5-24	0	7-5 - 7-38	0	8-32 - 8-45	0
6-1 - 6-3	0	7-39 - 7-40	1	8-46	1

The computers offered in this agreement, as well as the programs that TI has created to use with them, are tools that can help people better manage the information used in their business; but tools— including TI computers—cannot replace sound judgment nor make the manager's business decisions.

Consequently, TI cannot warrant that its systems are suitable for any specific customer application. The manager must rely on judgment of what is best for his or her business.

Printed in U.S.A.

Manual Update

MANUAL: DNOS Supervisor Call (SVC) Reference Manual (2270507-9701) *C

MCR/CHANGE NO.: MCR 004723/Change 1

EFFECTIVITY DATE: 20 March 1985

This change package contains information necessary to update your current manual. Please remove the obsolete pages from your existing manual and replace them with the changed pages as follows:

Remove Obsolete Pages	Insert Change 1 Pages
Cover/Manual Revision History	Cover/Effective Pages
—	Effective Pages Cont.
iii - iv	iii - iv
4-9 - 4-10	4-9 - 4-10
5-1 - 5-2	5-1 - 5-2
6-3 - 6-4	6-3 - 6-4
—	6-4A/6-4B
6-19 - 6-20	6-19 - 6-20
6-25 - 6-26	6-25 - 6-26
6-139 - 6-140	6-139 - 6-140
6-163 - 6-164	6-163 - 6-164
7-3 - 7-4	7-3 - 7-4
7-39 - 7-40	7-39 - 7-40
7-43 - 7-44	7-43 - 7-44
7-67 - 7-68	7-67 - 7-68
7-93 - 7-94	7-93 - 7-94
8-7 - 8-8	8-7 - 8-8
8-21 - 8-22	8-21 - 8-22
8-31 - 8-32	8-31 - 8-32
8-45 - 8-46	8-45 - 8-46
9-3 - 9-4	9-3 - 9-4
10-53 - 10-54	10-53 - 10-54
A-9 - A-10	A-9 - A-10
A-21 - A-22	A-21 - A-22
User's Resp./Bus. Reply	User's Resp./Bus. Reply
Inside Cover/Cover	Inside Cover/Cover

9 Volume Management — Describes the organization of disk volumes used with DNOS and describes in detail the SVCs that initialize, install, and unload volumes.

10 Task Support — Describes in detail the SVCs that support DNOS tasks by providing system services required during task execution.

11 System Interface — Describes in detail the SVCs that a task can use to obtain system data and to allocate and deallocate disk space.

12 SVC Compatibility — Describes in detail those SVCs that are supported by DNOS only to allow execution of programs for other operating systems. Specifies DNOS alternatives for more efficient processing under DNOS.

ppendix

A SVC Index — A special index of SVCs and SVC operations (sub-opcodes).

B Device Character Sets — Detailed descriptions in tabular form of character sets for the I/O devices supported by DNOS.

C Master/Slave Task Examples — Assembly listings of a typical owner task and requesting task for an IPC master/slave channel.

ı addition to this manual, the DNOS software manuals shown on the support manual diagram rontispiece) contain information related to DNOS SVCs.

he following manuals, not listed on the frontispiece, are referenced in this manual or contain ıformation related to DNOS SVCs.

Title	Part Number
Model 810 Printer Installation and Operation Manual	939460-9701
Model LP300 and LP600 Line Printers Installation and Operation Manual	2250364-9701
Model LQ45 Letter Quality Printer System Installation and Operation Manual	2268695-9701

Contents

Paragraph Title Page

1 — Introduction

2 — Job Management

3 — Program File Management

4 — Task Management

Paragraph Title Page

5 — Input/Output Operations

6 — Device I/O

LIST OF EFFECTIVE PAGES

INSERT LATEST CHANGED PAGES AND DISCARD SUPERSEDED PAGES

Note: The changes in the text are indicated by a change number at the bottom of the page and a vertical bar in the outer margin of the changed page. A change number at the bottom of the page but no change bar indicates either a deletion or a page layout change.

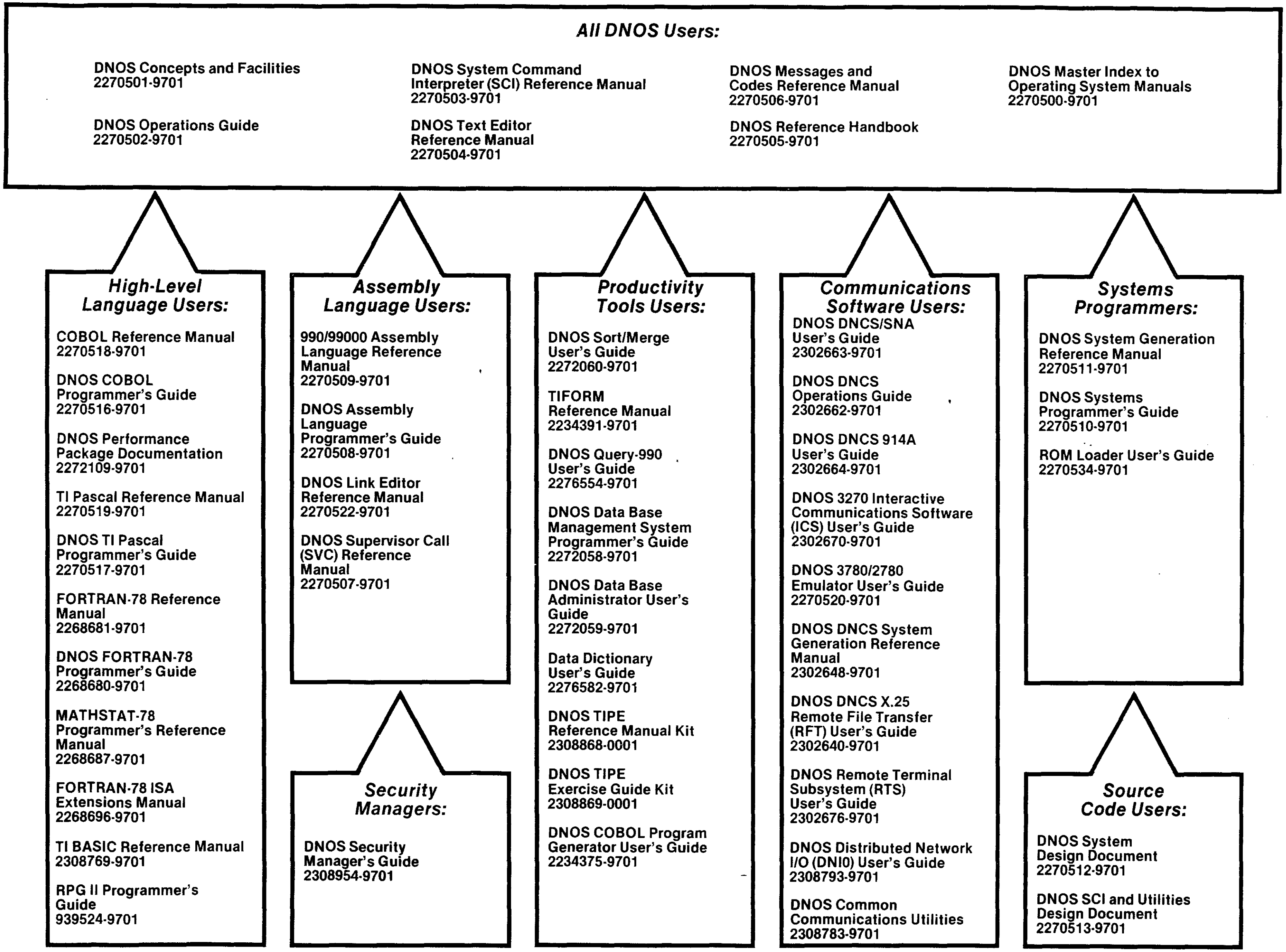
DNOS Supervisor Call (SVC) Reference Manual (2270507-9701)

Continued:

PAGE NO.	CHANGE NO.
8-47 - 8-56	0
9-1 - 9-3	0
9-4	1
9-5 - 9-8	0
10-1 - 10-53	0
10-54	1
10-55 - 10-68	0
11-1 - 11-12	0
12-1 - 12-6	0
A-1 - A-8	0
A-9	1
A-10 - A-20	0
A-21 - A-22	1
B-1 - B-38	0
C-1 - C-32	0
Index-1 - Index-8	0
User's Response	1
Business Reply	1
Inside Cover	1
Cover	1

DNOS Software Manuals

This diagram shows the manuals supporting DNOS, arranged according to user type. Refer to the block identified by your user group and all blocks above that set to determine which manuals are most beneficial to your needs.



All DNOS Users:

DNOS Concepts and Facilities
2270501-9701

DNOS Operations Guide
2270502-9701

DNOS System Command Interpreter (SCI) Reference Manual
2270503-9701

DNOS Text Editor Reference Manual
2270504-9701

DNOS Messages and Codes Reference Manual
2270506-9701

DNOS Reference Handbook
2270505-9701

DNOS Master Index to Operating System Manuals
2270500-9701

High-Level Language Users:

COBOL Reference Manual
2270518-9701

DNOS COBOL Programmer's Guide
2270516-9701

DNOS Performance Package Documentation
2272109-9701

TI Pascal Reference Manual
2270519-9701

DNOS TI Pascal Programmer's Guide
2270517-9701

FORTRAN-78 Reference Manual
2268681-9701

DNOS FORTRAN-78 Programmer's Guide
2268680-9701

MATHSTAT-78 Programmer's Reference Manual
2268687-9701

FORTRAN-78 ISA Extensions Manual
2268696-9701

TI BASIC Reference Manual
2308769-9701

RPG II Programmer's Guide
939524-9701

Assembly Language Users:

990/99000 Assembly Language Reference Manual
2270509-9701

DNOS Assembly Language Programmer's Guide
2270508-9701

DNOS Link Editor Reference Manual
2270522-9701

DNOS Supervisor Call (SVC) Reference Manual
2270507-9701

Security Managers:

DNOS Security Manager's Guide
2308954-9701

Productivity Tools Users:

DNOS Sort/Merge User's Guide
2272060-9701

TIFORM Reference Manual
2234391-9701

DNOS Query-990 User's Guide
2276554-9701

DNOS Data Base Management System Programmer's Guide
2272058-9701

DNOS Data Base Administrator User's Guide
2272059-9701

Data Dictionary User's Guide
2276582-9701

DNOS TIPE Reference Manual Kit
2308868-0001

DNOS TIPE Exercise Guide Kit
2308869-0001

DNOS COBOL Program Generator User's Guide
2234375-9701

Communications Software Users:

DNOS DNCS/SNA User's Guide
2302663-9701

DNOS DNCS Operations Guide
2302662-9701

DNOS DNCS 914A User's Guide
2302664-9701

DNOS 3270 Interactive Communications Software (ICS) User's Guide
2302670-9701

DNOS 3780/2780 Emulator User's Guide
2270520-9701

DNOS DNCS System Generation Reference Manual
2302648-9701

DNOS DNCS X.25 Remote File Transfer (RFT) User's Guide
2302640-9701

DNOS Remote Terminal Subsystem (RTS) User's Guide
2302676-9701

DNOS Distributed Network I/O (DNIO) User's Guide
2308793-9701

DNOS Common Communications Utilities
2308783-9701

Systems Programmers:

DNOS System Generation Reference Manual
2270511-9701

DNOS Systems Programmer's Guide
2270510-9701

ROM Loader User's Guide
2270534-9701

Source Code Users:

DNOS System Design Document
2270512-9701

DNOS SCI and Utilities Design Document
2270513-9701

DNOS Software Manuals Summary

Concepts and Facilities
Presents an overview of DNOS with topics grouped by operating system functions. All new users (or evaluators) of DNOS should read this manual.

DNOS Operations Guide
Explains fundamental operations for a DNOS system. Includes detailed instructions on how to use each device supported by DNOS.

System Command Interpreter (SCI) Reference Manual
Describes how to use SCI in both interactive and batch jobs. Describes command procedures and gives a detailed presentation of all SCI commands in alphabetical order for easy reference.

Text Editor Reference Manual
Explains how to use the Text Editor on DNOS and describes each of the editing commands.

Messages and Codes Reference Manual
Lists the error messages, informative messages, and error codes reported by DNOS.

DNOS Reference Handbook
Provides a summary of commonly used information for quick reference.

Master Index to Operating System Manuals
Contains a composite index to topics in the DNOS operating system manuals.

Programmer's Guides and Reference Manuals for Languages
Contain information about the languages supported by DNOS. Each programmer's guide covers operating system information relevant to the use of that language on DNOS. Each reference manual covers details of the language itself, including language syntax and programming considerations.

Performance Package Documentation
Describes the enhanced capabilities that the DNOS Performance Package provides on the Model 990/12 Computer and Business System 800.

Link Editor Reference Manual
Describes how to use the Link Editor on DNOS to combine separately generated object modules to form a single linked output.

Supervisor Call (SVC) Reference Manual
Presents detailed information about each DNOS supervisor call and DNOS services.

DNOS System Generation Reference Manual
Explains how to generate a DNOS system for your particular configuration and environment.

User's Guides for Productivity Tools
Describe the features, functions, and use of each productivity tool supported by DNOS.

User's Guides for Communications Software
Describe the features, functions, and use of the communications software available for execution under DNOS.

Systems Programmer's Guide
Discusses the DNOS subsystems and how to modify the system for specific application environments.

ROM Loader User's Guide
Explains how to load the operating system using the ROM loader and describes the error conditions.

DNOS Design Documents
Contain design information about the DNOS system, SCI, and the utilities.

DNOS Security Manager's Guide
Describes the file access security features available with DNOS.

Preface

This manual describes the supervisor calls (SVCs) supported by the DNOS operating system. In addition, the manual provides information about using SVCs and includes an example of a typical use of each SVC.

The intended audience of this manual is the programmer who writes SVCs and the associated call blocks in his program. The manual is assembly language oriented because the call block must be written in assembly language. Most SVCs required for high-level language programs are included in the run-time software and are transparent to the programmer. For those operations that require explicit SVCs, the high-level language programmer must refer to this manual for a detailed description of the supervisor call block. The DNOS language programmer's guide for each language describes the interface with the supervisor call block.

The sections and appendixes of this manual are organized as follows:

Section

1 Introduction — Provides a general description of a supervisor call, and a call block. Lists the conventions used in the manual.

2 Job Management — Describes the job concept and describes the operations of the Job Management SVC in detail.

3 Program File Management — Describes the program file on which the task, procedure, and program segments and overlays are stored. Describes in detail the SVCs used in program file management.

4 Task Management — Describes the DNOS task concept and describes in detail the SVCs used for controlling execution of tasks.

5 I/O Operations — Describes the supervisor call used to request I/O operations, and contains information common to all types of I/O.

6 Device I/O — Describes the I/O utility operations for device I/O and the I/O operations for each device, organized by device.

7 File I/O — Describes the I/O utility operations for file I/O and the I/O operations for each type of file, organized by file type.

8 Interprocess Communication — Describes the I/O utility operations for interprocess communication (IPC) and the I/O operations for symmetric and master/slave channels.

9 Volume Management — Describes the organization of disk volumes used with DNOS and describes in detail the SVCs that initialize, install, and unload volumes.

10 Task Support — Describes in detail the SVCs that support DNOS tasks by providing system services required during task execution.

11 System Interface — Describes in detail the SVCs that a task can use to obtain system data and to allocate and deallocate disk space.

12 SVC Compatibility — Describes in detail those SVCs that are supported by DNOS only to allow execution of programs for other operating systems. Specifies DNOS alternatives for more efficient processing under DNOS.

Appendix

A SVC Index — A special index of SVCs and SVC operations (sub-opcodes).

B Device Character Sets — Detailed descriptions in tabular form of character sets for the I/O devices supported by DNOS.

C Master/Slave Task Examples — Assembly listings of a typical owner task and requesting task for an IPC master/slave channel.

In addition to this manual, the DNOS software manuals shown on the support manual diagram (frontispiece) contain information related to DNOS SVCs.

The following manuals, not listed on the frontispiece, are referenced in this manual or contain information related to DNOS SVCs.

Title	Part Number
Model 810 Printer Installation and Operation Manual	939460-9701
Model LP300 and LP600 Line Printers Installation and Operation Manual	2250364-9701
Model LQ45 Letter Quality Printer System Installation and Operation Manual	2268695-9701

Contents

5 — Input/Output Operations

6 — Device I/O

7 — File I/O

8 — Interprocess Communication

9 — Volume Management

10 — Task Support

11 — System Interface

12 — SVC Compatibility

Appendixes

Index

Illustrations

Tables

1

Introduction

1.1 HOW TO USE THIS MANUAL

This manual contains information about the use of the supervisor calls (SVCs) of DNOS and a detailed description of each call. The manual also describes the supervisor call block required for each SVC.

The introductory material at the beginning of each section of the manual presents general information about the categories of supervisor calls supported by DNOS. Within each section, introductory paragraphs for the major headings provide information about the purpose and anticipated results of executing the SVCs described under the headings.

For each SVC, the manual includes both the general information needed to use the SVC and the detailed information required to write the code for the SVC. The manual is intended for reference purposes as the source of detailed information.

The supervisor call block is written in assembly language; the interface between a high-level language and assembly language for an SVC is described in the programmer's guide for that language.

This section includes information common to all SVCs and conventions used in the manual.

1.2 THE SUPERVISOR CALL

The interface between application programs and the DNOS operating system is the supervisor call. In programs written in a high-level language most supervisor calls are transparent to the user (provided in the run-time package for the language). However, the high-level language programmer may write SVCs to perform functions not otherwise available. The assembly language programmer must code all supervisor calls.

The supervisor call is implemented as an extended operation (XOP) in DNOS. Specifically, XOP 15 is the means of entry to the SVC processor of DNOS. The address supplied with the XOP instruction is that of the supervisor call block. The following is an example of the code for an SVC:

```
    BLK      DATA 0                 REPRESENTS A CALL BLOCK
```

The assembly language includes a directive that provides a convenient and meaningful substitute for the XOP command. The DXOP directive defines a symbolic operation code for an XOP. The following example defines SVC as XOP 15:

```
        DXOP      SVC,15                         DEFINES SVC AS XOP 15
```

Including the DXOP at the beginning of the program makes the following code valid for SVCs:

```
        SVC       @BLK                           EXECUTE SVC DEFINED IN BLOCK BLK
```

1.3 THE SUPERVISOR CALL BLOCK

The DNOS supervisor call block is the data structure that defines the supervisor call. The statements described in the preceding paragraph apply to all DNOS supervisor calls. The difference between SVCs is the content and format of the supervisor call block.

A supervisor call block consists of at least one byte and as many additional bytes as the SVC requires. The first byte (and only byte for some SVCs) contains the opcode that defines the SVC. Opcodes 0 through >7F are reserved for SVCs supported by DNOS.

NOTE

> Throughout this document a value preceded by a right angle bracket (>) indicates a hexadecimal value.

Table 1-1 lists the SVC opcodes. You may define SVCs for your applications in the range of >80 through >FF. Definition of SVCs is described in the *DNOS Systems Programmer's Guide*.

The second byte of many SVCs is the return code byte. DNOS returns a satisfactory completion code (zero in most cases) in this byte when the operation completes successfully. DNOS returns an error code in this byte when the operation completes in error. Error codes are listed and described in the *DNOS Messages and Codes Reference Manual.*

Several DNOS SVCs provide different operations as determined by a sub-opcode in the third byte of the supervisor call block. In these cases, the actual operation to be performed is selected by the opcode and the sub-opcode.

In some supervisor call blocks, designated bytes contain the result of the requested operation after the operation has completed. That is, the system returns values in some fields of some supervisor call blocks.

The additional bytes of supervisor call blocks may contain various types of information related to the operation, such as:

- Flags

- Input or output data

- Addresses of input or output data
- Size or count values
- Identifiers
- Task parameters
- LUNOs
- Character strings

The paragraph that describes each SVC also describes the associated supervisor call block. To locate the description of an SVC, look in the SVC index, Appendix A. The description includes the number of bytes required, whether or not the block must be aligned on a word (even) address, and the type of information in each byte.

Table 1-1. SVC Opcodes

SVC #	Name	Paragraph Number
00	I/O Operations	5.2.2, 5.3
	Resource-Independent Sub-opcodes: 00 — Open 01 — Close 02 — Close, Write EOF 03 — Open and Rewind 04 — Close and Unload 05 — Read Device Status 06 — Forward Space 07 — Backward Space 09 — Read ASCII 0B — Write ASCII 0D — Write EOF 0E — Rewind 0F — Unload	(These are described for each device. See Appendix A.)
	Many resource-specific sub-opcodes are supported.	(See Appendix A.)
01	Wait for I/O	5.3.1.1
02	Time Delay	4.4
03	Get Date and Time	10.7.1.1
04	End of Task	4.11
06	Suspend Task	4.7
07	Activate Suspended Task	4.8
09	Extend Time Slice	4.9
0A	Convert Binary to Decimal	10.2.1
0B	Convert Decimal to Binary	10.2.2
0C	Convert Binary to Hexadecimal	10.2.3
0D	Convert Hexadecimal to Binary	10.2.4
0E	Activate Time-Delayed Task	4.5
0F	Abort I/O Request by LUNO	5.3.2

Table 1-1. SVC Opcodes (Continued)

SVC #	Name	Paragraph Number
10	Get Common Data Address	12.2
11	Change Task Priority	4.6
12	Get Memory	10.5.1
13	Release Memory	10.5.2
14	Load Overlay	10.5.3
17	Get Task Bid Parameters	10.7.2
1B	Return Common Data Address	12.3
1C	Put Data	12.4
1D	Get Data	12.5
1F	Scheduled Bid Task	4.3
20	Install Disk Volume	9.3
21	System Log Queue Request	10.7.3
22	Disk Management	11.3
24	Suspend for Queue Input	11.4
25	Install Task Segment	3.2
26	Install Procedure/Program Segment	3.3
27	Install Overlay	3.4
28	Delete Task	3.5
29	Delete Procedure/Program Segment	3.6
2A	Delete Overlay	3.7
2B	Execute Task	4.2
2C	Read/Write TSB	11.5
2D	Read/Write Task	11.6
2E	Self-Identification	10.7.4
2F	Get End Action Status	10.7.5.1
31	Map Program Name to ID	3.9
33	Kill Task	4.10
34	Unload Disk Volume	9.4
35	Poll Status of Task	10.7.6
36	Wait for Any I/O	5.3.1.2
37	Assign Program File Space	3.8
38	Initialize New Disk Volume	9.2
3B	Set Date and Time	10.7.1.2
3D	Semaphore Operations	10.6.1
	Sub-opcodes:	
	00 — Signal	10.6.1.1
	01 — Wait	10.6.1.2
	02 — Test	10.6.1.3
	03 — Initialize	10.6.1.4
	04 — Modify	10.6.1.5
3E	Reset End Action Status	10.7.5.2
3F	Retrieve System Data	11.2
40	Segment Management	10.5.4

Table 1-1. SVC Opcodes (Continued)

SVC #	Name	Paragraph Number
	Sub-opcodes:	
	00 — Change Segment	10.5.4.1
	01 — Create Segment	10.5.4.2
	02 — Reserve Segment	10.5.4.3
	03 — Release Segment	10.5.4.4
	04 — Check Segment Status	10.5.4.5
	05 — Force Write Segment	10.5.4.6
	06 — Reserved	
	07 — Set/Reset Not Modified and Releasable	10.5.4.7
	09 — Load Segment	10.5.4.8
	0A — Unload Segment	10.5.4.9
	0B — Set Exclusive Use of a Segment	10.5.4.10
	0C — Reset Exclusive Use of a Segment	10.5.4.11
41	Initiate Event	10.6.2.1
42	Wait for Event	10.6.2.2
43	Name Management	5.2.1
	Sub-opcodes:	
	00 — Determine Name's Value	5.2.1.1
	02 — Set Name's Value	5.2.1.2
	04 — Delete Name	5.2.1.3
	0F — Restore Name	5.2.1.4
45	Get Encrypted Value	10.3.1
46	Get Decrypted Value	10.3.2
47	Log Accounting Entry	10.4.1
48	Job Management	2.2
	Sub-opcodes:	
	01 — Create Job	2.2.1
	02 — Halt Job	2.2.2
	03 — Resume Halted Job	2.2.2
	04 — Change Job Priority	2.2.4
	05 — Map Job Name to Job ID	2.2.5
	06 — Kill Executing Job	2.2.6
	07 — Delete Job	2.2.7
	09 — Get Job Information	2.2.8
49	Get Accounting Information	10.4.2
4C	Return Code Processor	10.7.7
4F	Post Event	10.6.2.3

1.4 SECURED SUPERVISOR CALLS

If your system uses the file security option, access rights must be granted before some SVCs can be performed. Table 1-2 shows the access rights required to execute these SVCs.

Table 1-2. Secured Supervisor Call Opcodes

SVC#	Name	Rights
>00	FILE I/O	
	SUB-OPCODES	
	>02 CLOSE-WRITE EOF	WRITE access to file
	>09 READ ASCII	READ access to file
	>0A READ DIRECT	READ access to file
	>0B WRITE ASCII	WRITE access to file
	>0C WRITE DIRECT	WRITE access to file
	>0D WRITE EOF	WRITE access to file
	>10 REWRITE	WRITE access to file
	>41 READ GREATER	READ access to file
	>42 READ BY KEY	READ access to file
	>44 READ EQUAL/GREATER	READ access to file
	>45 READ NEXT	READ access to file
	>46 INSERT	WRITE access to file
	>47 REWRITE	WRITE access to file
	>48 READ PREVIOUS	READ access to file
	>49 DELETE BY KEY/CURNT	WRITE access to file
	>59 MULTIPLE REC READ	READ access to file
	>5B MULTIPLE REC WRITE	WRITE access to file
	>91 ASSIGN LUNO	ANY access to file*
	>92 DELETE FILE	DELETE access to file
	>95 RENAME FILE	WRITE, DELETE access to file
	>96 UNPROTECT FILE	WRITE, DELETE access to file
	>97 WRITE PROTECT FILE	WRITE, DELETE access to file
	>98 DELETE PROTECT FILE	WRITE, DELETE access to file
>14	LOAD OVERLAY	EXECUTE access to program file
>1F	SCHEDULED BID TASK	EXECUTE access to program file
>25	INSTALL TASK	READ, WRITE access to program file
>26	INSTALL PROCEDURE	READ, WRITE access to program file
>26	INSTALL SEGMENT	READ, WRITE access to program file
>27	INSTALL OVERLAY	READ, WRITE access to program file
>28	DELETE TASK	READ, WRITE access to program file
>29	DELETE PROCEDURE	READ, WRITE access to program file
>29	DELETE SEGMENT	READ, WRITE access to program file
>2A	DELETE OVERLAY	READ, WRITE access to program file
>2B	BID TASK	EXECUTE access to program file
>31	MAP NAME TO ID	READ or EXECUTE access to program file
>37	ASSIGN PROG FILE SPACE	READ, WRITE access to program file
>40	SEGMENT MANAGEMENT	
	SUB-OPCODES	
	>00 CHANGE SEGMENT	Program segment-EXECUTE access to program file Relative record-READ, WRITE access to file
	>01 CREATE SEGMENT	Relative record-WRITE access to file
>43	NAME MANAGEMENT	
	SUB-OPCODES	
	>0F RESTORE NAMES	READ access to synonym file

Note:
* If the user has any access to the file, the assign LUNO will succeed.

1.5 ERROR CODES

The error codes returned in byte 1 of supervisor call blocks are listed in the *DNOS Messages and Codes Reference Manual*. The codes and corresponding messages for SVC errors are listed in the SVC error section of the manual. A table in that section lists the error codes in SVC opcode order, with the message number that corresponds to the error code. The numbered message identifies the error.

Many SVC processors call the I/O SVC, opcode 00, to perform I/O functions. When an I/O error occurs, the I/O SVC returns an error code of 00nn to the SVC processor. The SVC processor returns error nn as the error code. For example, the Scheduled Bid Task SVC, opcode >1F, issues an I/O SVC. If the I/O SVC returns error code >26 to the opcode >1F SVC processor, the SVC processor returns error code >1F26 to the calling task. This error code is not shown as >1F26 in the SVC error code table, but is shown as >0026; it is the same I/O error, whether the I/O SVC is called by the task directly or indirectly. Thus, when you receive an error code that is not in the table, change the first two hexadecimal digits to zeros and look again.

Often the goal in coding SVCs is to recognize that an error has occurred rather than to interpret every type of error. In some cases it is necessary to take appropriate action in the event of a specific error. In many cases it is adequate to display the error code.

1.6 CONVENTIONS

The following notational conventions are used in this manual:

Convention	Meaning
Greater than sign (>)	Identifies hexadecimal values.
Angle brackets (< >)	Enclose items returned to the supervisor call block.
Reserved	Designates a call block field or flag that must be set to zero.
[Reserved]	Designates a call block field that is reserved but may contain any value.

A set of conventions applies to the supervisor call block diagrams used in the manual. On the top line, at the left, the code and name of the SVC are printed. To the right, on the same line, the requirements and attributes of the SVC are printed. Additional lines are provided when several requirements and attributes apply. The requirements and attributes are:

- Align on word boundary
- Privileged tasks only
- System tasks only
- Can be initiated as an event

A requirement for many supervisor call blocks is that they must be aligned on word boundaries; that is, the first byte must have an even address. This may be accomplished by immediately preceding the first directive for the call block with an EVEN directive. However, if the first directive is a DATA directive, the call block is automatically aligned on a word boundary.

Two of the supervisor call attributes limit the use of the SVC to tasks having special attributes. A task may be installed as software privileged or as a system task. Those SVCs that have the attribute *privileged tasks only* may be executed only by software privileged tasks. SVCs that have the attribute *system tasks only* may be executed only by system tasks.

The Initiate Event SVC issues SVCs as events. The Wait for Event SVC allows a task to suspend itself pending completion of an SVC that has been initiated as an event. The SVCs that can be issued in this way have the attribute *can be initiated as an event*.

Two columns at the left of the diagram show the byte address of the left byte relative to the first byte of the block, in decimal and in hexadecimal. When a field of the block consists of more than two words, the first and last words are shown, with a break in the vertical lines to indicate that words have been omitted.

2

Job Management

2.1 JOB CONCEPT

A job is a collection of cooperating programs (called tasks in DNOS). Either a user or a job may initiate another job to perform one or more functions. A job may be an interactive job or a batch job. The tasks of an interactive job use a video display terminal (VDT) to communicate with the user. The tasks of a batch job do not communicate with the user.

A job has a set of attributes, which are associated with the job when it is initiated. The attributes of a job may become the attributes of a new job when the job initiates a new job. Optionally, attributes may be specified when the new job is created. Attributes include the user ID to identify the job with a user, and user-specified execution parameters. In this sense, the job represents the user by passing user information to the tasks of the job as required.

A job is also an environment of associated resources. These resources (for example, files, devices, and interprocess communications (IPC) channels) may be shared by the tasks of the job. Job-local LUNOs and job variables, including semaphores, synonyms, and logical names, are also shared by tasks. A job ID identifies this environment.

The priority of a job is specified when the job is created. It is one of the parameters used to determine the run-time priority of the tasks of the job.

There is always at least one job in the system: the system job. It is the first job in the sequence of creation of jobs; no hierarchy of jobs is maintained. It consists of a group of cooperating tasks that perform operating system functions, and the system resources are associated with the system job.

2.2 MANAGING JOBS

A task may request the DNOS job manager to perform the following functions:

- Create a job
- Halt execution of a job
- Resume execution of a job
- Modify priority of a job
- Return the job ID of a job
- Force abnormal termination of a job
- Return job information

In addition, tasks within the system job may request the DNOS job manager to:

- Delete a job
- Expand a job communications area (JCA)

The task executes a Job Management Request SVC (opcode >48) to access the job manager. The calling task is suspended during execution of the SVC. The Job Management Request SVC validates the requests for job manager services by performing security checks to prevent unauthorized access. Optionally, validation may be omitted.

The following operations may normally be performed only on jobs that have the same user ID as the current job:

- Halt execution of a job
- Resume execution of a job
- Return the job ID of a job
- Force abnormal termination of a job
- Return job information

One of the two exceptions to the restriction is that the system operator task may request these operations for any job. The other exception is that a task that was created with the *do not verify* flag (refer to the supervisor call block description) set to one may perform any of these operations for any job by setting the *do not verify* flag when requesting the operation. Only the system operator may modify the priority of a job.

None of the operations of the Job Management Request SVC may be performed on the system job.

Most of the operations of the Job Management Request SVC require the job run ID to identify the job. A task can obtain the job run ID of the job to which it belongs by executing either a Self-Identification SVC or a Get Job Information operation of the Job Management Request SVC. A task can obtain the job run ID of any job having the same user ID as the current job by executing a Map Job Name to Job ID operation. The task must have the job name to supply in the call block.

When a job has been initiated, a job state code may be returned by various status commands used with SCI. The job state codes and their meaning are listed in Table 2-1.

Table 2-1. Job State Codes

Code	Job State
01	Job is Being Created
02	Job is in an Executable State
03	Job is Halted
04	Job is Terminating
05	Job is Being Expanded

The supervisor call block for the Job Management Request SVC is as follows:

SVC >48 -- Job Management Request   Align on word boundary

Dec	Hex		
0	0	>48	<Return Code>
2	2	Sub-Opcode	Job Priority
4	4	Flags	
6	6	Job Run ID	
8	8	Job Name (8 Chars)	
14	E		
16	10	Initial Task ID	JCA Size
18	12	Task Bid Parameters (4 Bytes)	
20	14		
22	16	Task Station ID	Task Prog. File LUNO
24	18	Synonym Segment ID	
26	1A	Reserved	
28	1C	User ID (8 Characters)	
34	22		
36	24	User Password (8 Characters)	
42	2A		
44	2C	User Account Number (16 Characters)	
58	3A		

2279443

The call block contains the following:

Byte	Contents
0	Opcode, >48.
1	Return code. DNOS returns zero when the operation completes satisfactorily. When the operation completes in error, DNOS returns an error code.
2	Sub-opcode: >01 — Create Job. >02 — Halt Job. >03 — Resume Halted Job. >04 — Change Job Priority. >05 — Map Job Name to Job ID. >06 — Kill Executing Job. >07 — Delete Job (tasks of system jobs only). >09 — Get Job Information.
3	Job priority. A value in the range of 0 (highest priority) through 31 (lowest priority). Returned by the system for a Get Job Information operation.
4-5	Flags. Bit 0 — New user ID flag. Set as follows: 1 — Bytes 28-59 contain the user ID, password, and account number for the new job. 0 — The user ID, password, and account number of the current job apply to the new job. Bit 1 — Do not verify flag. This flag must have been set in the initiating job. Set as follows: 1 — Do not validate the request. 0 — Perform normal validation. Bit 2 — Batch flag. Set as follows: 1 — Batch job. No interaction with user. Tasks run at lower priority. 0 — Not a batch job. Bits 3-15 — Reserved.
6-7	Job run ID. The run-time ID of the job, returned by DNOS for Create Job, Map Job Name to Job ID, and Get Job Information operations. Supplied by the user task for other operations.
8-15	Job name. A job name consisting of no more than 8 characters, supplied by the user to identify the job. Returned by the system for a Get Job Information operation.
16	Initial task ID. The installed ID of the initial task. For a Get Job Information operation, the current task ID, returned by the system.

Byte	Contents
17	Job communications area (JCA) size. A value of 1 (smallest), 2 (medium), or 3 (largest). For a Get Job Information operation, user privilege level, returned by the system.
18-21	Task bid parameters. Bid parameters for the initial task.
22	Task station ID. Station ID for the initial task. If no station is associated with the initial task, specify >FF. When the field contains zero the calling task's associated station is used.
23	Task program file LUNO, or zero. Program file LUNO assigned to the program file on which the initial task is installed. When the field contains zero, the task is loaded from the system program file, .S$SHARED. When the field contains >FF, the task is loaded from the program file on which the calling task resides.
24-25	Synonym segment ID, or zero. The segment ID of the segment used for synonyms and logical names. When the field contains zero, no synonym segment is provided for the new job. The segment manager identifies a segment by the segment ID. More information on segment IDs is in paragraph 10.5.4.
26-27	Reserved. The value of this field is set to zero.
28-35	User ID. The identifier of the user in control of the job. Returned by the system for a Get Job Information operation.
36-43	User password. The password of the user in control of the job.
44-59	User account number. The number of the account to which the job charges are to be charged. Returned by the system for a Get Job Information operation.

2.2.1 Create Job

DNOS automatically creates the job for execution of user tasks and any additional jobs required to implement user commands. The Create Job operation is available to allow a user task to create a job if desired.

Sub-opcode >01 specifies the Create Job operation with which a task creates a new job. The operation creates the specified job, and executes the new job, unless the job limit has been reached. In that case, the new job is queued for execution when a job terminates.

A job is executed by placing a specified initial task into execution. The supervisor call block for a Create Job operation contains the task ID, the station ID, and task parameters for the initial task. These items correspond to those required by the Execute Task SVC.

The size of the JCA for a job is specified as 1, 2, or 3. Size 1 is the smallest JCA size, appropriate for jobs that contain few tasks, when the tasks execute serially and use few files. Size 2, the next larger JCA size, is appropriate for average jobs. Size 3, the largest JCA size, is appropriate for jobs having tasks executing in parallel.

When synonyms, logical names, or both may be used by the job, the synonyms and names must be initialized using the Snapshot Name Definitions (SND) command. SND writes the contents of the current job's synonyms and names to a disk file. If a program creates a job with a name segment, the segment must be restored to memory before the Create Job SVC is issued. The names can be restored using the Restore Name Segment operation of the Name Management SVC described in paragraph 5.2.1.4. The segment ID of the name segment is returned by the Restore Name Segment SVC and it must be placed into the Create Job SVC block. The Create Job SVC can then be issued.

NOTE

> Do not use the Set Name's Value operation (paragraph 5.2.1.2) to create a synonym or logical name segment.

All of the fields of the supervisor call block are used for the Create Job operation. The job run ID is returned by the system.

The value supplied in the job priority field is one of the factors that determines the run-time priorities of the tasks in the job. The range of values is 0 through 31. The value of 31 results in the lowest task priorities.

The job flags byte contains three flags to select three options. Set these flags as described in the next three paragraphs.

If the user of the new job is not the user of the current job, set the new user ID flag to 1 and supply the user ID in bytes 28 through 35, the user password in bytes 36 through 43, and the user account number in bytes 44 through 59.

To create a job in which the tasks may issue a Job Management Request SVC with the *do not verify* flag set to 1, set the *do not verify* flag to 1. The current job must also have been created with the *do not verify* flag set to 1, however. This flag is intended for use by the tasks of the operating system.

To create the new job as a batch job, set the batch flag to 1.

The user supplies a job name in the job name field. The job name identifies the job and may be used by tasks in a job to obtain the job ID (Map Job Name to Job ID operation).

The initial JCA size is determined by the value in the JCA size field. The JCA is expanded dynamically by the system; when the initial size is too small, the expansion may occur too frequently. When the initial size is too large, memory is not used efficiently. A value of 2 for the intermediate size is a good choice when in doubt.

Four fields of the call block relate to the initial task; these items are the same items required to place a task in execution using any SVC or System Command Interpreter (SCI) command. The initial task ID is the installed ID of the task, the ID under which it was installed on the program file. The task bid parameters consist of two words that may be accessed by the initial task using a Get Parameters SVC. The contents of these two words are determined by the requirements of the initial task. The task station ID is the ID of the station (terminal) with which the initial task is to be associated. The task program file LUNO is the global LUNO assigned to the program file that contains the initial task. When the task is in program file .S$SHARED, enter zero.

Use of the next two fields is optional. Each specifies the segment ID of an initialized segment. One initialized segment contains synonyms for the job; the other contains logical names for the job. The new job may not use synonyms unless an initialized synonym segment is specified; it may not use logical names unless an initialized logical name segment is specified. However, the new job may add either synonyms or logical names to the appropriate segment; it may also delete either synonyms or logical names as required.

The following is an example of coding for a supervisor call block for a Create Job operation:

```
         EVEN                    CREATE JOB DOITNOW AT PRIORITY 10
CREJOB   BYTE >48                WITH JCA SIZE 2. INITIAL TASK IS
CJERR    BYTE 0                  >3A ON PROGRAM FILE LUNO >2E AT
         BYTE >01                STATION 12. SYNONYM SEGMENT IS
         BYTE 10                 >2A. USER ID IS CSE010, PASSWORD
         DATA >8000              IS P7807, AND ACCOUNT NUMBER IS
         DATA 0                  263345426.
         TEXT 'DOITNOW'
         BYTE >3A
         BYTE 2
         DATA 0,0
         BYTE 12
         BYTE >2E
         DATA >2A
         DATA 0
         TEXT 'CSE010    '
         TEXT 'P7807     '
         TEXT '263345426'
```

2.2.2 Halt Job

The Halt Job operation suspends execution of the tasks of a job. Execution of the tasks resumes when a Resume Halted Job operation is performed.

Sub-opcode >02 specifies the Halt Job operation. Only the first 16 bytes of the supervisor call block apply. The specific fields are:

- Opcode
- Return code
- Sub-opcode
- Job run ID

A job may be halted if it has the same user ID as the current job, the current job was created with the *do not verify* flag set, or the current job is the system operator job.

The job run ID is supplied by the user task.

The following is an example of coding for a supervisor call block for a Halt Job operation:

```
           EVEN                    HALT JOB >4F.
HLTJOB     BYTE >48
HJERR      BYTE 0
           BYTE >02
           BYTE 0
           DATA 0
           DATA >4F
           DATA 0,0,0,0
```

2.2.3 Resume Halted Job

The Resume Halted Job operation restores execution of the tasks of a job when execution of the tasks was interrupted by a Halt Job operation.

Sub-opcode >03 specifies the Resume Halted Job operation. Only the first 16 bytes of the supervisor call block apply. The specific fields are:

- Opcode
- Return code
- Sub-opcode
- Job run ID

A job may be resumed if it has the same user ID as the current job. the current job was created with the *do not verify* flag set, or the current job is the system ope·ator job.

The job run ID is supplied by the user task.

The following is an example of coding for a supervisor call block for a Resume Halted Job operation:

```
           EVEN                    RESUME JOB >4F.
RESJOB     BYTE >48
RJERR      BYTE 0
           BYTE >03
           BYTE 0
           DATA 0
           DATA >4F
           DATA 0,0,0,0
```

2.2.4 Change Job Priority

The priority that is specified for a job when the job is created may be changed by this operation.

Sub-opcode >04 specifies the Change Job Priority operation. Only the first 16 bytes of the supervisor call block apply. The specific fields are:

- Opcode
- Return code
- Sub-opcode
- Job priority
- Job flags
- Job run ID

The *do not verify* flag is the only flag that applies to the Change Job Priority operation. When the current job was created with the *do not verify* flag set and the *do not verify* flag is set for the Change Job Priority operation, any task may change the job priority. Otherwise, only a task in the system operator job may change the job priority.

The job run ID is supplied by the user task.

The following is an example of coding for a supervisor call block for a Change Job Priority operation:

```
        EVEN                    CHANGE PRIORITY OF JOB >4F
CHJBPR  BYTE >48                TO 15
CPERR   BYTE 0
        BYTE >04
        BYTE 15
        DATA 0
        DATA >4F
        DATA 0,0,0,0
```

2.2.5 Map Job Name to Job ID

Any task may obtain the job ID that corresponds to a job name by performing a Map Job Name to Job ID operation.

Sub-opcode >05 specifies the Map Job Name to Job ID operation. Only the first 16 bytes of the supervisor call block apply. The specific fields are:

- Opcode
- Return code
- Sub-opcode
- <Job run ID>
- Job name

The job run ID is returned by the system. The job name is supplied by the user task.

The following is an example of coding for a supervisor call block for a Map Job Name to Job ID operation:

```
         EVEN                          OBTAIN JOB ID OF JOB
MAPJOB   BYTE >48                      REDALERT
MJERR    BYTE 0
         BYTE >05
         BYTE 0
         DATA 0
JBRID    DATA 0
         TEXT 'REDALERT'
```

The Map Job Name to Job ID operation may be issued for a job that has the same user ID as the current job, was created with the *do not verify* flag set, or is the system operator job. If two or more jobs have the same job name and a Map Job Name to Job ID operation is performed, an ID is returned along with a warning message.

2.2.6 Kill Executing Job

The Kill Executing Job operation forces abnormal termination of the tasks of a job. The job is effectively terminated when all tasks have terminated.

Sub-opcode >06 specifies the Kill Executing Job operation. Only the first 16 bytes of the supervisor call block apply. The specific fields are:

- Opcode
- Return code
- Sub-opcode
- Job run ID

A job may be killed by a job that has the same user ID as the current job, was created with the *do not verify* flag set, or is the system operator job.

The job run ID is supplied by the user task.

The following is an example of coding for a supervisor call block for a Kill Executing Job operation:

```
         EVEN                        KILL JOB >4F.
KILJOB   BYTE >48
KJERR    BYTE 0
         BYTE >06
         BYTE 0
         DATA 0
         DATA >4F
         DATA 0,0,0,0
```

2.2.7 Delete Job

The system uses the Delete Job operation to delete jobs that are no longer required. This operation is not available to user tasks.

Sub-opcode >07 specifies the Delete Job operation. Only the first 16 bytes of the supervisor call block apply. The specific fields are:

- Opcode
- Return code
- Sub-opcode
- Job run ID

The job run ID is supplied by the system task.

The following is an example of coding for a supervisor call block for a Delete Job operation:

```
         EVEN                        DELETE JOB >4F.
DELJOB   BYTE >48
DJERR    BYTE 0
         BYTE >07
         BYTE 0
         DATA 0
         DATA >4F
         DATA 0,0,0,0
```

2.2.8 Get Job Information

A task may obtain job information related to the job to which it belongs or for any job for which it can supply the job run ID. The job information returned consists of:

- Job run ID
- Job priority
- Job name
- Current task ID
- User ID
- User privilege level
- User account number

A job may get information for another job if it has the same user ID as the current job, was created with the *do not verify* flag set, or is the system operator job.

Sub-opcode >09 specifies the Get Job Information operation. The user supplies the job run ID or zero as the job run ID. When the user supplies zero, the information for the current job (the job to which the task belongs) is returned. The following fields of the supervisor call block apply:

- Opcode
- Return code
- Sub-opcode
- Job priority
- Job run ID
- Job name
- Initial task ID
- JCA size
- User ID
- User account number

The Get Job Information operation returns information in the supervisor call block. Much of the information is returned in the same fields in which it is supplied for a Create Job operation. However, two of the fields of the call block are redefined to return information that is not supplied for the Create Job operation.

The operation returns the job run ID in the job run ID field. When zero is supplied in this field, the returned ID is that of the current job. Otherwise the job run ID supplied in the field is returned.

The operation returns the job priority, job name, user ID, and user account number in the correspondingly designated fields of the call block.

The operation returns the user privilege level in the JCA size field, and the installed ID of the current task in the initial task ID field.

The following is an example of coding for a supervisor call block for a Get Job Information operation:

```
        EVEN                    GET JOB INFORMATION ON CURRENT JOB
GJINF   BYTE >48
GIERR   BYTE 0
        BYTE >09
JOBPRI  BYTE 0
        DATA 0
JBRTID  DATA 0
JBNAM   DATA 0,0,0,0
INITID  BYTE 0
UPRLEV  BYTE 0
        DATA 0,0
        BYTE 0
        BYTE 0
        DATA 0
        DATA 0
USERID  DATA 0,0,0,0
        DATA 0,0,0,0
USRACN  DATA 0,0,0,0,0,0,0,0
```

3

Program File Management

3.1 PROGRAM FILES

A program file is a relative record file that is organized to serve a special purpose. The purpose of a program file is to store the linked object code of task segments, procedure segments, program segments, and overlays in a format that allows these modules to be loaded into memory for execution.

The task concept is described more fully in Section 4; however, with respect to the program file, a task segment is the linked object module that contains the addresses required by DNOS to initiate execution of the task. It contains the workspaces and the data for the task. It may also include all or part of the executable code. The task segment is loaded into a program's memory space, or into one of the segments of a program's memory space, for execution.

A task may have one or two associated procedure segments in memory during its execution. These procedure segments contain reentrant executable code for the task. A procedure segment on a program file may be associated with more than one task. A procedure segment is loaded into an area of memory that is mapped into the memory space of each task that shares the procedure.

A program segment is used either as a procedure segment or as a data portion of a task. The management of program segments provides more flexibility than it provides for procedure segments. No specific content or function is defined for a program segment; it may be used as the program requires. When a program segment is disk-resident, it resides in a program file. The task executes a Segment Management SVC, described in Section 10, to access a program segment from the program file.

Alternately, one or two program segments can be associated with a task instead of either or both procedure segments. When this is done, DNOS loads the program segments along with the task segment, and they are available to the task without executing a Segment Management SVC.

An overlay is a linked object module that is loaded from a program file while the task is executing, often replacing a previously loaded overlay. Overlays are loaded into the task memory space at the designated address, and do not require a special or additional segment. Overlays can be associated with task segments and program segments.

The modules in the program file are written in blocks that correspond to file records. The program file contains a directory record that lists information for each module in the file. The information associates the ID and name with the numbers of the records that contain the module. The directory also lists the characteristics of the modules.

The task segment and overlays associated with that task segment must all be in the same program file. The procedure segments for a task must be either in program file .S$SHARED, or in the same program file as the task segment. In the program file .S$SHARED some IDs are reserved for the exclusive use of Texas Instruments software products. The hexadecimal values of the reserved IDs range from 00 to 0F for tasks and from 00 to 2F for procedure segments.

When a task segment is installed, it may be installed as a hardware privileged task. A hardware privileged task may contain the privileged instructions of the computer.

Another installation option is software privilege. A software privileged task is allowed to execute privileged SVCs. For example, the Install Task SVC can be executed only by a software privileged task.

A task may be installed as a system task. A system task is a task whose task memory area is mapped with system memory.

This section consists of descriptions of the supervisor calls that install and delete all three types of program modules. The SVC that loads an overlay, the SVC that assigns space on a program file, and the SVC that maps a program name to an ID are described also.

3.2 INSTALLING A TASK SEGMENT

A task may install a task segment in a program file by executing an Install Task Segment SVC (opcode >25). The SVC processor writes the module to the program file and writes the appropriate entry in the program file directory. The following options are available:

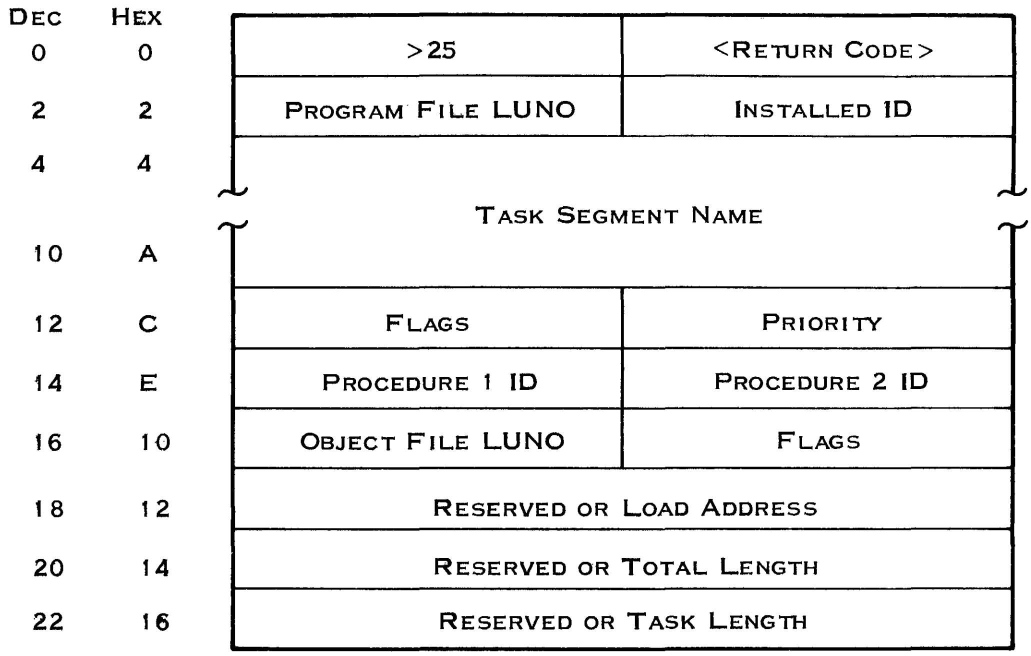
- The task segment may be installed in the system program file.
- The task segment ID (installed ID) may be automatically assigned.
- The task segment may be installed as a hardware privileged task.
- The task segment may be installed as a software privileged task.
- The task segment may be installed as a system task.
- The task segment may be installed as memory resident. (The task does not actually become memory resident until the next IPL. It may be executed from the disk until the next IPL.)
- The task segment may be installed as delete protected.
- The task segment may be installed as replicatable.

- The task segment may be installed as execute protected in the Model 990/12 Computer.
- The task segment may be installed as one that takes end action on an arithmetic overflow in the Model 990/12 Computer.
- The task segment may be installed as one that uses the writable control store in the Model 990/12 Computer.

The supervisor call block for the Install Task Segment SVC is as follows:

SVC >25 -- INSTALL TASK SEGMENT

ALIGN ON WORD BOUNDARY
PRIVILEGED TASKS ONLY



DEC	HEX		
0	0	>25	<RETURN CODE>
2	2	PROGRAM FILE LUNO	INSTALLED ID
4	4	TASK SEGMENT NAME	
10	A		
12	C	FLAGS	PRIORITY
14	E	PROCEDURE 1 ID	PROCEDURE 2 ID
16	10	OBJECT FILE LUNO	FLAGS
18	12	RESERVED OR LOAD ADDRESS	
20	14	RESERVED OR TOTAL LENGTH	
22	16	RESERVED OR TASK LENGTH	

2279444

The call block contains the following:

Byte	Contents
0	Opcode, >25.
1	Return code. DNOS returns zero when the operation completes satisfactorily. When the operation completes in error, DNOS returns an error code.
2	Program file LUNO, or zero. The LUNO assigned to the program file. When this field contains zero, DNOS installs the task segment on .S$SHARED. When this field contains >FF, the system installs the task on the program file on which the calling task resides. If the special install flag is set to one, the LUNO must be open for exclusive write. Otherwise, the LUNO must be closed.

Byte **Contents**

3 Installed task segment ID, or zero. When this field contains zero, DNOS assigns an available ID and returns the ID in this field. The user may specify the installed ID in this byte. Do not assign a reserved system task ID to a task installed on the system program file.

4–11 Name of task segment, or zero. When this field contains zero, DNOS uses the IDT in the object module as the task segment name. The task segment name consists of not more than eight alphanumeric characters, the first of which must be alphabetic. The name is left justified in the field, filled to the right with spaces.

12 Flags. (Byte 17 also contains flags.)
Bit 0 — Privileged task flag. Set as follows:
  1 — Task may execute privileged assembly language instructions.
  0 — Task may not execute privileged assembly language instructions.
Bit 1 — System task flag. Set as follows:
  1 — System memory is mapped into the task memory area.
  0 — System memory is not mapped into the task memory area.
Bit 2 — Memory resident flag. Set as follows:
  1 — Task is installed as memory resident.
  0 — Task is installed as disk resident.
Bit 3 — Delete protected flag. Set as follows:
  1 — Task segment may not be deleted until this flag is set to zero (refer to the SCI command, Modify Task Entry).
  0 — Task segment may be deleted.
Bit 4 — Replicatable flag. Set as follows:
  1 — Copies of the task may be loaded into memory and executed.
  0 — Only one copy of the task may be loaded into memory and executed.
Bit 5 — Procedure segment 1 program file flag. Set as follows:
  1 — Procedure segment 1 is on .S$SHARED.
  0 — Procedure segment 1 is on the program file on which the task segment is being installed.
Bit 6 — Procedure segment 2 program file flag. Set as follows:
  1 — Procedure segment 2 is on .S$SHARED.
  0 — Procedure segment 2 is on the program file on which the task segment is being installed.
Bit 7 — Special install flag. Set to zero. The flag is set to one only by the system to install a task segment when the image is already on the program file. Installation consists only of writing the file directory entry; the task segment is not replaced.

13 Priority. The installed priority of the task, 0 through 4, or real-time priority, 0 through 127. Bit 0 of the field is set to one for a real-time priority.

14 Procedure segment 1 ID. The ID of the procedure segment or program segment to be attached as procedure segment 1, or zero when there is no attached procedure segment.

Byte **Contents**

15 Procedure segment 2 ID. The ID of the procedure segment or program segment to be attached as procedure segment 2, or zero when no procedure is to be attached as procedure segment 2. A procedure segment 2 is valid only if procedure segment 1 is specified.

16 Object file LUNO. The LUNO assigned to the object file. This LUNO must not be open.

17 Flags. (Byte 12 also contains flags.)
Bit 0 — Overflow flag. Set as follows:
 1 — Task transfers control to end action routine when an arithmetic overflow occurs (Model 990/12 Computer only).
 0 — Task does not transfer control to end action when an arithmetic overflow occurs.
Bit 1 — Writable control store flag. Set as follows:
 1 — Task uses writable control store (Model 990/12 Computer only).
 0 — Task does not use writable control store.
Bit 2 — Execute protect flag. Set as follows:
 1 — Set hardware execute protection (Model 990/12 Computer only). Execute protection prohibits instruction accesses to the memory in which the task segment is loaded.
 0 — Do not set hardware execute protection.
Bit 3 — Software privileged.
 1 — Task will be allowed to issue privileged SVCs.
 0 — Task will not be allowed to issue privileged SVCs.
Bit 4 — Updateable.
 1 — Task segment will be rewritten to the program file by the segment manager if modified.
 0 — Task segment will not be rewritten to the program file by the segment manager.
Bit 5 — Reusable.
 1 — Task segment may be reused consecutively without reloading.
 0 — Task segment must be reloaded for each use.
Bit 6 — Copyable.
 1 — Task segment may be replicated by copying an in-memory copy.
 0 — Task segment may be replicated only by copying the disk-resident copy.
Bit 7 — Reserved.

18-19 Reserved. When special install flag is set to one, the load address of the task segment.

20-21 Reserved. When special install flag is set to one, the length of the task segment.

22-23 Reserved. When special install flag is set to one, the length of the portion of the task segment that is not overlaid.

The following is an example of coding for a supervisor call block for an Install Task Segment SVC:

```
        EVEN             INSTALL TASK SEGMENT ON PROGRAM
ITASK   BYTE >25         FILE ASSIGNED TO LUNO >2C (LUNO IS
ITERR   BYTE 0           NOT OPEN). TASK ID IS >83, TASK
        BYTE >2C         NAME IS MYTASK. TASK IS PRIVILEGED
        BYTE >83         AND REPLICATABLE, HAS NO PROCEDURES.
        TEXT 'MYTASK     INSTALL AT PRIORITY 2.
        BYTE >88         OBJECT FILE LUNO IS >4A.
        BYTE 2
        DATA 0
        BYTE >4A
        BYTE 0
        DATA 0
        DATA 0
        DATA 0
```

3.3 INSTALLING A PROCEDURE SEGMENT OR PROGRAM SEGMENT

A task may install a procedure segment or program segment in a program file by executing an Install Procedure/Program Segment SVC (opcode >26). The SVC writes the segment to the program file and writes the appropriate entry in the program file directory. The following options are available:

- The procedure segment or program segment may be installed as memory resident. (The procedure segment or program segment does not actually become memory resident until the next IPL. It may be executed from the disk until the next IPL.)

- The procedure segment or program segment may be installed as delete protected.

- The procedure segment or program segment may be installed as execute protected in the Model 990/12 Computer.

- The procedure segment or program segment may be installed to use the writable control store in the Model 990/12 Computer.

- The procedure segment or program segment may be installed as write protected in the Model 990/12 Computer.

The supervisor call block for the SVC is as follows:

SVC >26 -- INSTALL PROCEDURE/PROGRAM SEGMENT

ALIGN ON WORD BOUNDARY
PRIVILEGED TASKS ONLY

DEC	HEX		
0	0	>26	<RETURN CODE>
2	2	PROGRAM FILE LUNO	SEGMENT ID
4	4	PROCEDURE SEGMENT NAME	
10	A		
12	C	FLAGS	OBJECT FILE LUNO
14	E	RESERVED OR LOAD ADDRESS	
16	10	RESERVED OR SEGMENT LENGTH	
18	12	FLAGS	RESERVED

2279445

The call block contains the following:

Byte	Contents
0	Opcode, >26.
1	Return code. DNOS returns zero when the operation completes satisfactorily. When the operation completes in error, DNOS returns an error code.
2	Program file LUNO, or zero. The LUNO assigned to the program file. When this field contains zero, install procedure segment or program segment on program file .S$SHARED. When this field contains >FF, install the segment on the program file on which the calling task resides. If the special install flag is set to one, the LUNO must be open for exclusive write. Otherwise, the LUNO must be closed.
3	Procedure or program segment ID, or zero. When this field contains zero, DNOS assigns an available ID and returns the ID in this field. The user may specify the procedure or program segment ID in this byte.

Byte **Contents**

4-11 Name of procedure segment, or zero. When this field contains zero, DNOS uses the IDT in the object module as the procedure name. The name consists of not more than eight alphanumeric characters, the first of which must be alphabetic. The name is left justified in the field, filled to the right with spaces.

12 Flags. (Byte 18 contains flags which apply to a program segment only.)

Bit 0 — Segment flag. Set as follows:
- 1 — Install a program segment.
- 0 — Install a procedure segment.

Bit 1 — For a procedure segment, reserved. For a program segment, system flag. Set as follows:
- 1 — Program segment may only be accessed by system tasks.
- 0 — Program segment may be accessed by all tasks.

Bit 2 — Memory resident flag. Set as follows:
- 1 — Procedure segment or program segment is installed as memory resident.
- 0 — Procedure segment or program segment is installed as disk resident.

Bit 3 — Delete protected flag. Set as follows:
- 1 — Procedure segment or program segment may not be deleted until this flag is set to zero (refer to the Modify Procedure Segment Entry (MPE) command in the *DNOS SCI Reference Manual*).
- 0 — Procedure segment or program segment may be deleted.

Bit 4 — For a procedure segment, writable control store flag. Set as follows:
- 1 — Task uses writable control store (Model 990/12 Computer only).
- 0 — Task does not use writable control store.

For a program segment, replicatable flag. Set as follows:
- 1 — Program segment is replicatable.
- 0 — Program segment is not replicatable.

Bit 5 — For a procedure segment, execute protect flag. Set as follows:
- 1 — Set hardware execute protection (Model 990/12 Computer only). Execute protection prohibits instruction accesses to the memory in which the task is loaded.
- 0 — Do not set hardware execute protection.

For a program segment, share protected flag. Set as follows:
- 1 — Program segment may not be shared concurrently.
- 0 — Program segment may be shared.

Bit 6 — For a procedure segment, write protect flag. Set as follows:
- 1 — Set hardware write protection (Model 990/12 Computer only). Write protection prohibits write accesses to the memory in which the procedure or segment is loaded.
- 0 — Do not set hardware write protection.

For a program segment, reserved.

Byte **Contents**

Bit 7 — For a procedure or program segment, the special install flag. Set to zero. The flag is set to one only by the system to install a procedure segment when the image is already on the program file. Installation consists only of writing the file directory entry.

13 Object file LUNO. The LUNO assigned to the object file. This LUNO must not be open.
Bit 0 — Segment flag. Set as follows:
1 — Install a program segment.
0 — Install a procedure segment.

14-15 Reserved. When the special install flag (bit 7 of byte 12) is set to one, the load address of the procedure segment.
Bit 0 — Segment flag. Set as follows:
1 — Install a program segment.
0 — Install a procedure segment.

16-17 Reserved. When the special install flag (bit 7 of byte 12) is set to one, the length of the procedure segment.

18 Flags. For a procedure segment, set to zero. For a program segment, set as follows:
Bit 0 — Reserved. Set to zero.
Bit 1 — Writable control store.
1 — Program segment accesses writable control store (Model 990/12 Computer only).
0 — Program segment does not access writable control store.
Bit 2 — Execute protect.
1 — Set hardware execute protection for program segment (Model 990/12 Computer only).
0 — Do not set hardware execute protection.
Bit 3 — Write protect.
1 — Set hardware write protection for program segment (Model 990/12 Computer only).
0 — Do not set hardware write protection.
Bit 4 — Updateable.
1 — Program segment will be rewritten to the program file by the segment manager if modified.
0 — Program segment will not be rewritten to the program file by the segment manager.
Bit 5 — Reusable.
1 — Program segment may be used consecutively without reloading.
0 — Program segment must be reloaded for each access.

Bit 6 — Copyable.
1 — Program segment may be replicated by copying an in-memory copy.
0 — Program segment may be replicated only by copying the disk-resident copy.
Bit 7 — Reserved.

19 Reserved.

The following is an example of coding for a supervisor call block for an Install Procedure/Segment SVC:

```
         EVEN             INSTALL PROCEDURE ON PROGRAM FILE
IPROC    BYTE >26         ASSIGNED TO LUNO >AA (LUNO IS NOT
IPERR    BYTE 0           OPEN). PROCEDURE ID IS ASSIGNED
         BYTE >AA         BY THE SYSTEM. OBJECT FILE LUNO
PROCID   BYTE 0           IS >BB. PROCEDURE NAME IS
         TEXT 'MYPROC     MYPROC
         BYTE 0
         BYTE >BB
         DATA 0
         DATA 0
         BYTE 0
         BYTE 0
```

3.4 INSTALLING AN OVERLAY

A task may install an overlay in a program file by executing an Install Overlay SVC (opcode >27). The SVC writes the overlay module to the program file and writes the appropriate entry in the program file directory. The following options are available:

- The overlay may be installed as relocatable

- The overlay may be installed as delete protected

The supervisor call block for the SVC is as follows:

SVC >27 -- INSTALL OVERLAY

ALIGN ON WORD BOUNDARY
PRIVILEGED TASKS ONLY

DEC	HEX		
0	0	>27	<RETURN CODE>
2	2	PROGRAM FILE LUNO	OVERLAY ID
4	4	OVERLAY NAME	
10	A		
12	C	FLAGS	ASSOCIATED SEGMENT ID
14	E	OBJECT FILE LUNO	RESERVED
16	10	RESERVED OR LOAD ADDRESS	
18	12	RESERVED OR OVERLAY LENGTH	
20	14	RESERVED	

2279446

The call block contains the following:

Byte	Contents
0	Opcode, >27.
1	Return code. DNOS returns zero when the operation completes satisfactorily. When the operation completes in error, DNOS returns an error code.
2	Program file LUNO, or zero. The LUNO assigned to the program file. When this field contains zero, install overlay on .S$SHARED. When this field contains >FF, install overlay on the program file on which the calling task resides. If special install flag is set to one, LUNO must be open for exclusive write. Otherwise, LUNO must be closed.
3	Overlay ID, or zero. When this field contains zero, DNOS assigns an available ID, and returns the ID in this field. The user may specify the overlay ID in this byte.

Byte	Contents
4–11	Name of overlay, or zero. When this field contains zero, DNOS uses the IDT in the object module as the overlay name. The name consists of not more than eight alphanumeric characters, the first of which must be alphabetic. The name is left justified in the field, filled to the right with spaces.
12	Flags. Bit 0 — Relocation flag. Set as follows: 1 — Relocation of addresses in the overlay is performed when the overlay is loaded. 0 — The overlay is to be loaded without relocation. Bits 1-2 — Reserved. Bit 3 — Delete protected flag. Set as follows: 1 — Overlay may not be deleted until this flag is set to zero (refer to the Modify Overlay Entry (MOE) command in the DNOS System Command Reference (SCI) Reference Manual). 0 — Overlay may be deleted. Bits 4-5 — Reserved. Bit 6 — Task/Segment flag. Set as follows: 1 — Associated segment ID is a program segment ID. 0 — Associated segment ID is a task segment ID. Bit 7 — Special install flag. Set to zero. The flag is set to one only by the system to install an overlay when the program image is already on the program file. Installation consists of writing the file directory entry.
13	Associated segment ID. Installed ID of the segment associated with the overlay. Overlay and segment must be on the same program file. Overlay is deleted automatically when the segment is deleted.
14	Object file LUNO. The LUNO assigned to the object file. This LUNO must not be open.
15	Reserved.
16–17	Reserved. When special install flag is set to one, load address of overlay.
18–19	Reserved. When special install flag is set to one, length of overlay in bytes.
20–21	Reserved.

The following is an example of coding for a supervisor call block for an Install Overlay SVC:

```
        EVEN              INSTALL OVERLAY ON PROGRAM FILE
IOVLY   BYTE >27          ASSIGNED TO LUNO >CA (LUNO IS NOT
IOERR   BYTE 0            OPEN). OVERLAY ID IS ASSIGNED
        BYTE >CA          BY THE SYSTEM. OVERLAY NAME IS
OVLID   BYTE 0            OLAY1. OBJECT FILE LUNO IS >0A.
        TEXT 'OLAY1       OVERLAY IS RELOCATABLE AND
        BYTE >80          IS ASSOCIATED WITH TASK >83.
        BYTE >83
        DATA >0A
        DATA 0
        DATA 0
        DATA 0
```

3.5 DELETING A TASK

A task may delete a task from a program file by executing a Delete Task SVC (opcode >28). The LUNO assigned to the program file may be open or closed. When the task is delete protected, it is not deleted. Associated overlays are also deleted unless they are delete protected.

The supervisor call block for the SVC is as follows:

SVC >28 -- DELETE TASK

ALIGN ON WORD BOUNDARY
PRIVILEGED TASK ONLY

DEC	HEX		
0	0	>28	<RETURN CODE>
2	2	PROGRAM FILE LUNO	INSTALLED ID
4	4	FLAGS	RESERVED

2279448

The call block contains the following:

Byte	Contents
0	Opcode, >28.
1	Return code. DNOS returns zero when the operation completes satisfactorily. When the operation completes in error, DNOS returns an error code.
2	Program file LUNO. The LUNO assigned to the program file. The LUNO must be closed unless the open flag is set to one. Enter zero when the task is installed on .S$SHARED. Enter >FF when the task and the calling task are installed on the same program file.

Byte	Contents
3	Installed ID of task to be deleted. Task must not be delete protected.
4	Flags. Bits 0-6 — Reserved. Bit 7 — Open flag. Set as follows: 1 — Program file LUNO is open. 0 — Program file LUNO is not open.
5	Reserved.

The following is an example of coding for a supervisor call block for a Delete Task SVC:

```
         EVEN                       DELETE TASK >83 ON PROGRAM FILE
DELTSK   BYTE >28                   LUNO >4A. LUNO IS NOT OPEN.
DELTER   BYTE 0
         BYTE >4A
         BYTE >83
         DATA 0
```

3.6 DELETING A PROCEDURE SEGMENT OR PROGRAM SEGMENT

A task may delete a procedure segment or program segment from a program file by executing a Delete Procedure/Program Segment SVC (opcode >29). The LUNO assigned to the program file may be open or closed. When the procedure segment or program segment is delete protected, it is not deleted. If the segment is a program segment with associated overlays, the overlays are also deleted if they are not delete protected.

The supervisor call block for the SVC is as follows:

SVC >29 -- DELETE PROCEDURE/PROGRAM SEGMENT

ALIGN ON WORD BOUNDARY
PRIVILEGED TASK ONLY

DEC	HEX		
0	0	>29	<RETURN CODE>
2	2	PROGRAM FILE LUNO	SEGMENT ID
4	4	FLAGS	RESERVED

2279449

The call block contains the following:

Byte	Contents
0	Opcode, >29.
1	Return code. DNOS returns zero when the operation completes satisfactorily. When the operation completes in error, DNOS returns an error code.
2	Program file LUNO. The LUNO assigned to the program file. The LUNO must be closed unless the open flag is set to one. Enter zero when the segment is installed on .S$SHARED or >FF when the segment is installed on the same program file as the executing task.
3	Segment ID of procedure segment or program segment to be deleted. Segment must not be delete protected.
4	Flags. Bits 0-6 — Reserved. Bit 7 — Open flag. Set as follows: 1 — Program file LUNO is open. 0 — Program file LUNO is not open.
5	Reserved.

The following is an example of coding for a supervisor call block for a Delete Procedure/Program Segment SVC:

```
        EVEN                DELETE PROCEDURE >35 ON PROGRAM
DELPR   BYTE >29            FILE LUNO >4F. THE LUNO IS NOT
DELPER  BYTE 0              OPEN.
        BYTE >4F
        BYTE >35
        DATA 0
```

3.7 DELETING AN OVERLAY

A task may delete an overlay from a program file by executing a Delete Overlay SVC (opcode >2A). The LUNO assigned to the program file may be open or closed. When the overlay is delete protected, it is not deleted.

The supervisor call block for the SVC is as follows:

SVC >2A -- DELETE OVERLAY

ALIGN ON WORD BOUNDARY
PRIVILEGED TASK ONLY

DEC	HEX		
0	0	>2A	<RETURN CODE>
2	2	PROGRAM FILE LUNO	OVERLAY ID
4	4	FLAGS	RESERVED

2279450

The call block contains the following:

Byte **Contents**

0 Opcode, >2A.

1 Return code. DNOS returns zero when the operation completes satisfactorily. When the operation completes in error, DNOS returns an error code.

2 Program file LUNO. The LUNO assigned to the program file. The LUNO must be closed unless the open flag is set to one. Enter zero when the overlay is installed on .S$SHARED or >FF when the overlay is installed on the same program file as the executing task.

3 Overlay ID of overlay to be deleted. Overlay must not be delete protected.

4 Flags.
Bits 0-6 — Reserved.
Bit 7 — Open flag. Set as follows:
1 — Program file LUNO is open.
0 — Program file LUNO is not open.

5 Reserved.

The following is an example of coding for a supervisor call block for a Delete Overlay SVC:

```
         EVEN                      DELETE OVERLAY >3B ON PROGRAM
DELOVL   BYTE >2A                  FILE LUNO >1F. LUNO IS NOT OPEN.
DLOLER   BYTE 0
         BYTE >1F
         BYTE >3B
         DATA 0
```

3.8 ASSIGNING SPACE ON A PROGRAM FILE

A task may request the assignment of a starting record in a program file by executing an Assign Program File Space SVC (opcode >37). A task, procedure, segment, or overlay module may then be written on the program file at the assigned starting record. This SVC is normally used by system utilities.

The supervisor call block for the SVC is as follows:

SVC >37 -- ASSIGN PROGRAM FILE SPACE

ALIGN ON WORD BOUNDARY
PRIVILEGED TASK ONLY

DEC	HEX		
0	0	>37	<RETURN CODE>
2	2	PROGRAM FILE LUNO	RESERVED
4	4	LENGTH	
6	6	<RECORD NUMBER>	
8	8	RESERVED	

2279451

The call block contains the following:

Byte	Contents
0	Opcode, >37.
1	Return code. DNOS returns zero when the operation completes satisfactorily. When the operation completes in error, DNOS returns an error code.
2	Program file LUNO. The LUNO assigned to the program file. Enter zero when the program file is .S$SHARED, or >FF when the program file is the program file on which the calling task resides. The LUNO must be open with exclusive write access privileges.

Byte	Contents
3	Reserved.
4-5	Length. Length in bytes of the module to be written.
6-7	Record number. DNOS returns the record number of the starting record of the assigned disk space.
8-9	Reserved.

The following is an example of coding for a supervisor call block for an Assign Program File Space SVC:

```
        EVEN               ASSIGN SPACE FOR 100 BYTES ON
APRFSP  BYTE >37           PROGRAM FILE LUNO >2C
APFSER  BYTE 0
        BYTE >2C
        BYTE 0
        DATA 100
        DATA 0
        DATA 0
```

3.9 MAPPING A PROGRAM NAME TO AN ID

A task may obtain the name or the installed ID of a task segment, procedure segment, or overlay, when either of the two items is known. The Map Program Name to ID SVC (opcode >31) returns the name or installed ID in the supervisor call block. A LUNO must be assigned to the program file that contains the specified module. Flags in the block specify the desired operation.

The supervisor call block for the SVC is as follows:

SVC >31 -- MAP PROGRAM NAME TO ID — ALIGN ON WORD BOUNDARY

DEC	HEX		
0	0	>31	<RETURN CODE>
2	2	FLAGS	[RESERVED]
4	4	PROCEDURE, TASK, OR OVERLAY NAME OR <PROCEDURE, TASK, OR OVERLAY NAME>	
10	A		
12	C	LUNO	<ID> OR ID
14	E	RESERVED	

2279452

The call block contains the following:

Byte **Contents**

0 Opcode, >31.

1 Return code. DNOS returns zero when the operation completes satisfactorily. When the operation completes in error, DNOS returns an error code.

2 User flags.
Bits 0-1 contain type of segment:
  00 — Task segment.
  01 — Procedure segment.
  10 — Overlay.
Bit 2 defines the operation:
  1 — Return program name.
  0 — Return program ID.
Bits 3-6 [Reserved].
Bit 7 specifies current state of LUNO:
  1 — LUNO opened by this task.
  0 — LUNO not opened by this task. This bit must be zero to map IDs for LUNOs >00 and >FF.

3 [Reserved].

4-11 Name of module (eight alphanumeric ASCII characters). Supplied by task if flag bit 2 is 0; returned by system if flag bit 2 is 1.

12 LUNO assigned to the program file that contains the module. A zero in this field specifies .S$SHARED, and >FF specifies the program file on which the calling task resides.

13 Segment ID. Returned by system if flag bit 2 is 0; supplied by task if flag bit 2 is 1.

14-15 Reserved.

The following is an example of coding for a supervisor call block for a Map Program Name to ID SVC that requests the name of a task segment:

```
        EVEN             GET NAME OF TASK >1B ON PROGRAM
MPIDNA  BYTE >31         FILE ASSIGNED TO LUNO >2C (LUNO IS
MPNERR  BYTE 0           NOT OPEN).
FLAGA   BYTE >20
        BYTE 0
NAME    BSS 8
LNUM    BYTE >2C
TID     BYTE >1B
        DATA 0
```

Another example of a supervisor call block for a Map Program Name to ID SVC returns a procedure ID:

```
         EVEN              GET ID OF PROCEDURE CALCFOUR ON
MPNAID   BYTE >31          PROGRAM FILE ASSIGNED TO LUNO >23
MPNERC   BYTE 0            (LUNO IS NOT OPEN).
FLAGS    BYTE >40
         BYTE 0
PNAM     TEXT 'CALCFOUR'
LUNO     BYTE >23
PRID     BYTE 0
         DATA 0
```

4

Task Management

4.1 TASK CONCEPT

A task, in the DNOS context, is a program executing under DNOS. Each task consists of an address space. The CPU context of the task defines the task at any given time. The context consists of a workspace, the address of which is in the workspace pointer register; an instruction address, in the program counter; and a status, stored in the status register.

The address space of the task may consist of one or more segments; no more than three segments are accessible at a given time. Segments of an executing task may be dynamically exchanged for other segments as required. (See the paragraph on memory control in the section on task support.)

When a task is installed in a program file on disk, it is designated as replicatable or nonreplicatable. A replicatable task is a task that may be loaded into memory and placed in execution when one or more copies of the task already exist in memory and are being executed. A nonreplicatable task may not be loaded into memory when a copy of the task is in memory being executed. When a nonreplicatable task is being executed, another user must wait until execution completes before executing that task.

Each task is assigned a priority level when it is installed. The highest priority level, 0, is reserved for system tasks. Real-time priority levels R1 through R127 follow priority level 0. Next in descending order of priority is priority level 1, intended for interactive tasks. It is followed by priority level 2, adequate for multiple disk accesses. The lowest priority, level 3, is automatically assigned to background tasks. Priority level 4 is the floating priority level and ranges between levels 1 and 2.

To optimize system performance, DNOS assigns run-time priorities dynamically, based on the installed priority (as modified by the Change Task Priority SVC) and on other factors.

Each task is installed on a program file with an installed ID unique with respect to the other tasks on the file. The run-time ID is assigned to the task when it is executed. Each task can obtain its run-time ID by executing a Self-Identification SVC. When a task issues an Execute Task SVC to execute another task, the system returns the run-time ID of the called task. If the calling task needs to have the called task know the run-time ID of the calling task, the calling task must pass its own run-time ID to the called task as a task parameter.

A task becomes a ready task when it has been loaded into memory and has been placed on a queue for execution. A ready task is awaiting its turn to be the executing task (in control of the CPU). The executing task may place itself in time delay, suspend itself, or request an operation (I/O, for example) that suspends the task while awaiting completion of the operation. Otherwise, the executing task completes its time slice and again becomes a ready task.

4.2 EXECUTING A TASK

One task may initiate execution of another task by issuing an Execute Task SVC (opcode >2B). The called task must have been installed on a program file. If the called task is not in execution, it is loaded and executed. If it is a replicatable task already in execution, another copy of the task is placed in execution. If the task shares a procedure segment with other tasks, the replication of the task also shares the procedure segment.

The following options are supported:

- The called task may execute in the foreground of an interactive job and alternate with the System Command Interpreter (SCI).
- The called task may execute in the background of either an interactive or batch job.
- The called task may execute in an interactive job under control of the system debug utility.
- The called task may be unconditionally suspended when it is ready to execute.
- The calling task may terminate following execution of this SVC.
- The calling task may be suspended until the called task has terminated.
- The called task may be associated with a specified job other than the job of the calling task (calling task must have been installed as a software privileged task). This option should only be used by system tasks.

The supervisor call block for the Execute Task SVC is as follows:

SVC >2B -- EXECUTE TASK — ALIGN ON WORD BOUNDARY

DEC	HEX		
0	0	>2B	<RETURN CODE>
2	2	INSTALLED ID <RUN ID>	FLAGS
4	4	PARAMETER 1	
6	6	PARAMETER 2	
8	8	STATION ID	PROGRAM FILE LUNO
10	A	JOB ID	

2279453

The call block contains the following:

Byte **Contents**

0 Opcode, >2B.

1 Return code. DNOS returns zero when the operation completes successfully. When the operation completes in error, DNOS returns an error code.

2 Installed ID of the task. DNOS returns the run-time ID in this byte.

3 Flags.
Bit 0 — Job ID flag. May be set by software privileged tasks only. Set as follows:
  1 — Job ID specified in bytes 10–11.
  0 — Same job as calling task.
Bit 1 — Reserved.
Bit 2 — Reserved.
Bit 3 — Background task flag. Set as follows:
  1 — Called task is to execute in background.
  0 — Called task is not a background task.
Bit 4 — Terminate flag. Set as follows:
  1 — Terminate calling task immediately (following successful initiation of execution of called task). All other user flags are ignored.
  0 — Do not terminate calling task.
Bit 5 — Reserved.
Bit 6 — Unconditional suspend flag. Set as follows:
  1 — Suspend the called task unconditionally after loading (typically used by the system debug utility).
  0 — Execute the called task after loading.
Bit 7 — Calling task suspend flag. Set as follows:
  1 — Suspend the calling task until the called task has terminated.
  0 — Do not suspend the calling task.

4-7 Task parameters. Enter parameters required by the called task, if any. Called task executes a Get Parameters SVC to access these parameters.

8 Station ID. Enter numeric portion of station ID for the station (terminal) with which called task is to be associated. Enter 0 when called task is to be associated with the same station as the calling task. Enter >FF when the task is not to be associated with a station.

9 Program file LUNO. The LUNO assigned to the program file on which the called task is installed. Enter zero when the task is installed on the S$SHARED program file. Enter >FF when the calling and called tasks are on the same program file.

10–11 Job ID. The ID of the job with which the task is associated. Applies only when bit 0 of flag byte (byte 3) is set to one.

For clarification of the use of the terminate flag feature, an example follows. If task A bids task B using the Execute Task SVC with bit 7 set in the supervisor call block, then task A is suspended in state >17 until task B terminates. However, if task B then bids task C with the terminate flag set in the Execute Task supervisor call block, task A is reactivated only when task C terminates. Similarly, if task C then bids task D with the terminate flag set in the Execute Task supervisor call block, task A is reactivated only when task D terminates. In other words, task A is reactivated only when the last task in the task chain has terminated. In most applications, task A is SCI.

The following is an example of coding for a supervisor call block for an Execute Task SVC:

```
         EVEN             EXECUTE TASK >3B ON FILE ASSIGNED TO
EXTSK    BYTE >2B         LUNO >1A; NO ASSOCIATED STATION,
         BYTE 0           NOT BACKGROUND TASK,
         BYTE >3B         EXECUTE IMMEDIATELY, SUSPEND
         BYTE >01         CALLING TASK. PASS FOUR CHARACTERS
         TEXT 'HELP'      AS PARAMETERS. SAME JOB ID AS
         BYTE >FF         CALLING TASK.
         BYTE >1A
         DATA 0
```

4.3 SCHEDULING A TASK

A task may schedule itself to resume execution at a specified time and date, or it may schedule another task to execute at a specified time and date. The Scheduled Bid Task SVC (opcode >1F) schedules execution of a task. When the called task is also the calling task, it is suspended. It resumes execution at the instruction following the SVC call at the specified time and date. When the calling task calls another task, control returns to the calling task following execution of the SVC. The scheduled time should be at least several seconds later than the time the SVC is executed to allow for execution of this SVC. Normally, the called task is installed on the system program file and is associated with the same station (terminal) and job as the calling task.

The following options are supported:

- The program file LUNO and a different station ID may be specified.
- The called task may be associated with a different job than the calling task (calling task must be software privileged).

The supervisor call block for the Scheduled Bid Task SVC is as follows:

SVC >1F -- SCHEDULED BID TASK ALIGN ON WORD BOUNDARY

DEC	HEX		
0	0	>1F	<RETURN CODE>
2	2	INSTALLED ID	YEAR
4	4	DAY	
6	6	HOUR	MINUTE
8	8	SECOND	FLAGS
10	A	PARAMETER 1	
12	C	PARAMETER 2	
14	E	STATION ID	PROGRAM FILE LUNO
16	10	JOB ID	

2279454

The call block contains the following:

Byte	**Contents**
0	Opcode, >1F.
1	Return code. DNOS returns zero when the operation completes satisfactorily. When the operation completes in error, DNOS returns an error code.
2	Installed ID of the task.
3	Year. The two least significant digits of the year in which the task is to be executed; example: 80.
4-5	Day of the year. The ordinal (Julian) date; example: 42 for February 11.
6	Hour. Range of 0 (midnight) through 23; example: 13 for 1 P.M.
7	Minute. Range of 0 through 59.
8	Second. Range of 0 through 59.
9	Flags. Bit 0 — Station ID/LUNO flag. Set as follows: 1 — Station ID and program file LUNO specified in bytes 14 and 15. 0 — Use station ID of calling task and system program file. Bit 1 — Job ID flag. May be set by privileged tasks only. Set as follows: 1 — Job ID specified in bytes 16-17. 0 — Same job as calling task. Bits 2-7 — Reserved.
10-13	Task parameters. Enter parameters required by the called task, if any. Called task executes a Get Parameters SVC to access these parameters.
14	Station ID. Enter numeric portion of station ID for station (terminal) with which called task is to be associated. Enter >FF when the task is not to be associated with a station.
15	Program file LUNO. The LUNO assigned to the program file on which the called task is installed. When the program file is .S$SHARED, enter zero. Enter >FF when the called and calling tasks are on the same program file.
16-17	Job ID. The ID of the job with which the task is associated.

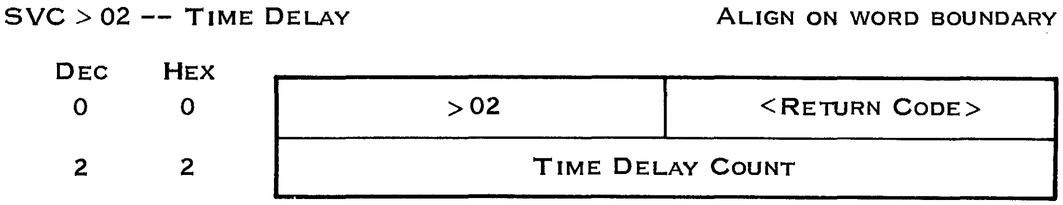
The following is an example of coding for a supervisor call block for a Scheduled Bid Task SVC:

```
         EVEN           EXECUTE TASK >2A ON SYSTEM PROGRAM
SBTSK    BYTE >1F       FILE ASSOCIATED WITH STATION AND
         BYTE 0         JOB OF CALLING TASK AT 5PM
         BYTE >2A       MAR. 21, 1980. PASS FOUR
         BYTE 80        CHARACTERS AS PARAMETERS.
         DATA 81
         BYTE 17
         BYTE 0
         BYTE 0
         BYTE 0
         TEXT 'OKAY'
         BYTE 0
         BYTE 0
         DATA 0
```

4.4 DELAYING TASK EXECUTION

A task may place itself in a time delay for a specified number of 50-millisecond periods. The Time Delay SVC (opcode >02) suspends the calling task until the specified time has elapsed or until another task executes an Activate Time Delay Task SVC that specifies the run-time ID of the time delayed task. When a cooperating task is to terminate the delay period, the time delayed task must communicate its run-time ID to the cooperating task before entering the time delay. The requested delay period is specified as a multiple of 50 milliseconds; it is a minimum delay period. That is, a count of zero provides a delay of from 0 to 50 milliseconds; a count of one provides a delay of from 50 to 100 milliseconds.

The supervisor call block for the Time Delay SVC is as follows:



SVC > 02 -- TIME DELAY   ALIGN ON WORD BOUNDARY

DEC	HEX		
0	0	>02	<RETURN CODE>
2	2	TIME DELAY COUNT	

2279455

The call block contains the following:

Byte	Contents
0	Opcode, >02.
1	Return code. DNOS returns zero when the operation completes satisfactorily. When the operation completes in error, DNOS returns an error code.
2-3	Time delay count. The number of 50-ms periods for the minimum delay.

The following is an example of coding for a supervisor call block for a Time Delay SVC:

```
          EVEN                           SUSPEND FOR 10 SECONDS
TIMEDL    BYTE >02
          BYTE 0
          DATA 200
```

4.5 RESUMING EXECUTION OF DELAYED TASK

A task may restore execution of another task that is in the time delay state. The Activate Time Delay Task SVC (opcode >0E) changes the state of a specified task from time delay to ready. The task to be activated is identified by its run-time ID, which must have been sent to the calling task by the time delay task prior to initiating the delay.

The supervisor call block for the Activate Time Delay Task SVC is as follows:

SVC >0E -- ACTIVATE TIME DELAY TASK

DEC	HEX		
0	0	>0E	<TASK STATE CODE>
2	2	RUN-TIME ID	

2279456

The call block contains the following:

Byte	Contents
0	Opcode, >0E.
1	Task-state code. DNOS returns the task-state code. Normal completion state is >05. When the task is not in the system, the code returned is >FF. Otherwise, when the task is not in time delay, the code is one of the following, defined in the table of task-state codes: >00->04, >06->4C
2	Run-time ID of task to be activated.

The following is an example of coding for a supervisor call block for an Activate Time Delay Task SVC:

```
ACTDLT    BYTE >0E                       ACTIVATE TIME DELAY TASK
          BYTE 0
RTID      BYTE 0                         RUN-TIME ID MOVED TO THIS BYTE
```

4.6 CHANGING PRIORITY OF A TASK

A task may change its priority level while executing. This capability is useful for a low priority task that contains a high priority operation. The Change Task Priority SVC (opcode >11) changes the priority of the calling task to that specified in the supervisor call block. The real-time priorities, R1 through R127, are identified by setting the most significant bit of the byte that contains the priority to 1.

The supervisor call block for the Change Task Priority SVC is as follows:

SVC >11 -- CHANGE TASK PRIORITY

DEC	HEX		
0	0	>11	NEW PRIORITY LEVEL <OLD PRIORITY LEVEL>

2279457

The call block contains the following:

Byte	Contents
0	Opcode, >11.
1	New priority level for task. Bit 0 is set to 1 to identify a real-time priority level. DNOS returns the old priority level in this byte, or error code >80 is returned when the requested priority level is not valid.

The following is an example of coding for a supervisor call block for a Change Task Priority SVC:

```
CTPL      BYTE >11             CHANGE PRIORITY OF CALLING
          BYTE >D0             TASK TO R50
```

4.7 SUSPENDING A TASK UNCONDITIONALLY

A task may unconditionally suspend itself while executing. The task remains suspended until a cooperating task executes an Activate Suspended Task SVC that specifies the run-time ID of the suspended task.

The Unconditional Suspend SVC (opcode >06) suspends the calling task unconditionally. When a cooperating task has executed an Activate Suspended Task SVC for the calling task prior to execution of the Unconditional Suspend SVC, the calling task is suspended and immediately reactivated.

The supervisor call block for the Unconditional Suspend SVC is as follows:

SVC >06 -- UNCONDITIONAL SUSPEND

DEC	HEX	
0	0	>06

2279458

The call block contains the following:

Byte	Contents
0	Opcode, >06.

The following is an example of coding for a supervisor call block for an Unconditional Suspend SVC:

```
UNCSUS    BYTE >06                      UNCONDITIONAL SUSPEND
```

4.8 ACTIVATING A SUSPENDED TASK

A task may activate another task that has suspended itself. The Activate Suspended Task SVC (opcode >07) activates a suspended task specified by its run-time ID. The suspended task must have communicated its run-time ID to the calling task before suspending itself. When the specified task is not yet suspended, it resumes execution immediately after executing an Unconditional Suspend SVC.

The supervisor call block for the Activate Suspended Task SVC is as follows:

SVC > 07 -- ACTIVATE SUSPENDED TASK

DEC	HEX		
0	0	>07	<TASK STATE CODE>
2	2	RUN-TIME ID	

2279459

The call block contains the following:

Byte **Contents**

0 Opcode, >07.

1 Task-state code. DNOS returns the task-state code. Normal completion state is >06. When the task is not in the system, the code returned is >FF. Otherwise, when the task is not suspended, the code is one of the following, defined in the table of task-state codes.

>00->05, >07->4C

When the task is not suspended and it executes an Unconditional Suspend SVC, it resumes execution immediately.

2 Run-time ID. The run-time ID of the suspended task to be activated.

The following is an example of coding for a supervisor call block for an Activate Suspended Task SVC:

```
ACTTSK    BYTE >07                 ACTIVATE SUSPENDED TASK
          BYTE 0
RTTID     BYTE 0                   RUN-TIME ID MOVED TO THIS BYTE
```

If the suspended task was loaded via the XHT procedure, the Activate Suspended Task SVC will appear to complete successfully but will actually have no effect. The Activate Suspended Task SVC processor checks for a task being debugged and does not activate tasks with the halted flag set.

4.9 INHIBITING TASK PREEMPTION

A task may extend its time slice, inhibiting the normal preemption that occurs at the end of a time slice. The SVC also inhibits preemption by a higher priority task. This allows the task to complete a critical function without interference by the system or by another task. The Extend Time Slice SVC (opcode >09) inhibits the system from preempting the task for a specified number of 50-ms periods. The task may suspend itself during the specified period by executing a Wait for I/O SVC, a Time Delay SVC, an Unconditional Suspend SVC, or an SVC (I/O SVC, for example) that suspends the task awaiting completion of an operation.

The supervisor call block for the Extend Time Slice SVC is as follows:

SVC >09 -- EXTEND TIME SLICE

DEC	HEX		
0	0	>09	TIME UNIT COUNT

2279460

The call block contains the following:

Byte	Contents
0	Opcode, >09.
1	Time unit count. The number of time units (50 milliseconds each) to extend the time slice. A count of zero provides a 200-millisecond extension of the time slice.

The following is an example of coding for a supervisor call block for an Extend Time Slice SVC:

```
EXTISL    BYTE >09                  EXTEND TIME SLICE 50 TIME
          BYTE 50                   UNITS — 2.5 SECONDS.
```

4.10 FORCING ABNORMAL TERMINATION

A task may force abnormal termination of another task in the same job. The Kill Task SVC (opcode >33) forces termination of a task having the specified run-time ID. The station ID of the station with which the task is associated may be specified also. When no station ID is supplied, the task is terminated whether or not it is associated with a station. When a station ID is supplied, the task is not terminated unless it is associated with the station. When >FF is supplied as the station ID, the task is terminated only if it is not associated with a terminal. Placing the appropriate value in the station ID field provides a degree of protection against terminating the wrong task. The Kill Task SVC completes as soon as DNOS has begun the termination or has put the task into its end action routine.

The supervisor call block for the Kill Task SVC is as follows:

SVC >33 -- KILL TASK ALIGN ON WORD BOUNDARY

DEC	HEX		
0	0	>33	<RETURN CODE>
2	2	RUN-TIME ID	STATION ID
4	4	<TASK STATE>	RESERVED
6	6	RESERVED	

2279461

The call block contains the following:

Byte **Contents**

0 Opcode, >33.

1 Return code. DNOS returns zero when the operation completes successfully. When the operation completes in error, DNOS returns an error code.

2 Run-time ID. Run-time ID of the task to be killed.

3 Station ID. Set to 0, the numeric portion of the ID of the station with which the task is associated, or to >FF. When set to 0, DNOS kills the task without regard to whether or not it is associated with a terminal. When set to a station ID, DNOS kills the task only if it is associated with the specified terminal. When set to >FF, DNOS kills the task only if it is not associated with a terminal.

4 Task state. DNOS returns the task state code of the terminated task in this byte.

5-7 Reserved.

The following is an example of coding for a supervisor call block for a Kill Task SVC:

```
          EVEN                      KILL TASK ASSOCIATED WITH
KILTSK    BYTE >33                  STATION 03
          BYTE 0
RUNID     BYTE 0                    RUN-TIME ID MOVED HERE
          BYTE 3
TSKST     BYTE 0
          BYTE 0,0,0
```

4.11 TERMINATING A TASK

The End of Task SVC (opcode >04) is the normal termination of a task. The SVC releases the local logical device tables and performs other termination functions. For disk-resident tasks, the SVC releases task memory and the task status block (TSB). For memory-resident tasks, the SVC reinitializes the TSB and clears outstanding breakpoints.

The supervisor call block for the End of Task SVC is as follows:

SVC > 04 -- END OF TASK

DEC	HEX	
0	0	>04

2279462

The call block contains the following:

Byte	Contents
0	Opcode, >04.

The following is an example of coding for a supervisor call block for an End of Task SVC:

```
ENDTSK    BYTE >04                  END OF TASK
```

5

Input/Output Operations

5.1 INPUT/OUTPUT CAPABILITIES

DNOS supports I/O operations to various types of devices and to several types of files. In addition, DNOS supports communication between programs, in which each program is analogous to a peripheral device of the other. To include all types of I/O, this manual refers to devices, files, and communication channels between programs as I/O resources.

This section contains information that applies to input and output (I/O) to devices, files, and interprocess communication (IPC) channels. Section 6 discusses device I/O in detail. Section 7 discusses file I/O in detail. Section 8 discusses IPC in detail.

DNOS supports two concepts of I/O to resources. Many I/O operations apply to various resources and are essentially the same for each resource. This concept is called resource-independent I/O. Resource-independent I/O allows the programmer to code I/O for any of several resources. A logical unit number (LUNO) represents the resource. The association of a specific resource with an I/O operation is made at run time by assigning a LUNO to the appropriate resource. The program may use different peripherals each time it is executed. Resource-independent I/O always accesses data sequentially, and is supported for terminals, magnetic tape units, line printers, card readers, sequential files, and communication channels.

With resource-independent I/O, operation of the resource is restricted to a mode that is common to several other resources. Resource-specific I/O allows the programming of specific capabilities of the device. Like resource-independent I/O, resource-specific I/O is directed to a LUNO rather than to the resource, but the LUNO must be assigned to the proper resource. Resource-specific I/O, which allows random access to data, is supported for terminals, relative record files, key indexed files, and communication channels.

Both resource-independent and resource-specific I/O may be requested in the initiate mode, under control of the initiate flag in the user flags field of the call block. In the initiate mode, control returns to the calling task after the operation is initiated; otherwise, the task is suspended during the entire I/O operation. Only three concurrent operations in the initiate mode per task per logical unit number (LUNO) are allowed. Additional operations cause the task to be suspended until completion whether the initiate mode is requested or not.

DNOS supports ANSI standard access modes to assist intertask I/O synchronization. The modes grant access privileges to a resource through a logical unit and deny conflicting accesses using other logical units. The Open, Open Rewind, and Open Random operations enforce access privileges. The operation fails if the requested access conflicts with existing access privileges. An Open operation is required for access to any I/O resource. The Modify Access Privileges operation changes access privileges for sequential and relative record files. For devices, change the access privileges by executing another Open operation requesting the desired privileges.

In order for the Open operations to enforce access privileges, these operations (not the Assign LUNO operation) actually associate the LUNO with the device, file, or IPC channel.

With respect to an access privilege, a Write operation is any operation that transfers data to a device, alters the contents of a file, or transfers data from a program to a communication channel.

- Read only — Allows calling program to read but not write. Allows read-only, shared, and exclusive write access by other programs.
- Shared — Allows calling program to read and to rewrite. Allows read-only and shared access by other programs.
- Exclusive write — Allows calling program to read and write. Allows read-only access by other programs.
- Exclusive all — Allows calling program to read and write. Allows no access by other programs.

Table 5-1 shows the accesses allowed when a program has opened a resource with each type of access. The column headings of the table refer to types of access for which a program has already opened the resource. The rows of the table refer to types of access by other programs.

Table 5-1. Access Mode Compatibility

	Resource Opened for			
Proposed Access	**Read Only***	**Shared***	**Exclusive Write***	**Exclusive All***
Read Only	A	A	A	F
Shared	A	A	F	F
Exclusive Write	A	F	F	F
Exclusive All	F	F	F	F

Note:

* A indicates allowed; F indicates forbidden.

5.2 PREPARING FOR I/O

Every I/O resource is identified for access purposes by a pathname. A pathname identifies a device, a file, or an IPC channel.

The pathname of a file consists of a volume name (which may be implied), directory names (if any), and a final component, which identifies a file. The names within the pathname are separated by periods (.). The pathname can contain a maximum of 48 characters. When the volume name is that of the system disk, it may be omitted. The pathname begins with a period in this case. The number of directory names in the pathname depends upon the organization of the disk. The volume directory and directories at all levels may contain both directories and files. The pathname of an IPC channel is similar to a file pathname.

When DNOS is used in a Business System computer network, a pathname can also include the site name for a computer in the network. The pathname for a file or device at a particular site includes the site name (followed by a colon) then the standard form for the local pathname.

A logical name may be used within a job to represent a pathname or a portion of a pathname. It consists of no more than eight characters and is considerably easier to use than a pathname with several elements. A logical name may have a set of parameters associated with it, which may be accessed by the tasks in the job. File characteristics, access privileges, and file creation parameters are examples of the types of information supplied as logical name parameters.

A logical name is also used to provide logical concatenation of files, or multivolume files. A set of pathnames is represented by a logical name. I/O to a LUNO assigned to this logical name is performed to the logical concatenation of the files. The pathnames in the set may be on more than one volume to provide multivolume files.

Logical names may be created using System Command Interpreter (SCI) commands or supervisor calls. However, the use of logical names in I/O operations is optional. Logical names are required for logically concatenated files and for resources that require parameters.

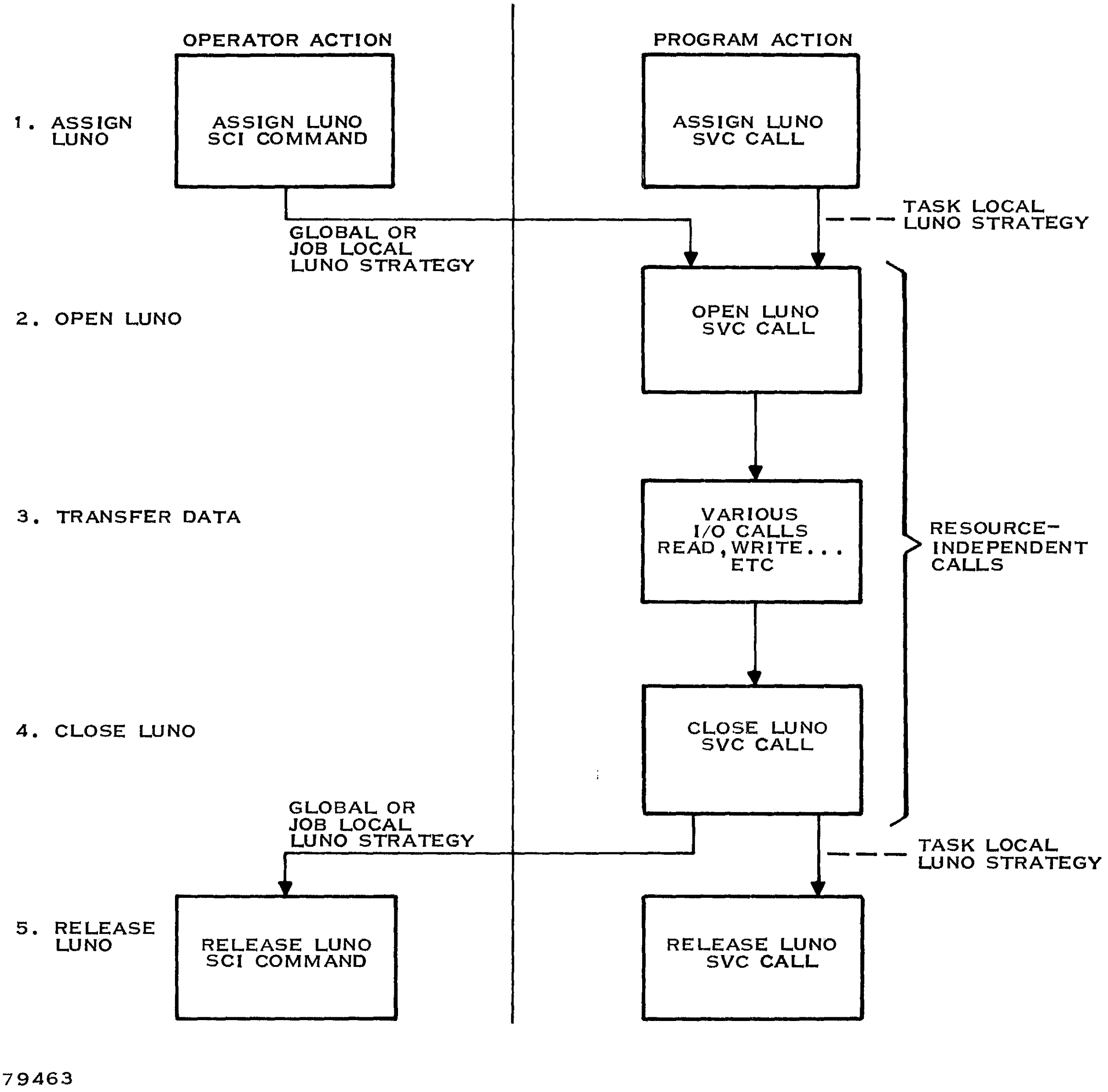
A LUNO must be assigned to a device name, logical name, or combination of logical name and pathname to associate the device, file, or IPC channel represented by the name with the LUNO. A LUNO may be assigned using either an SCI command or an SVC. Assigning LUNOs by command at run time allows different resources to be assigned for each run. On the other hand, assigning LUNOs with SVCs is more convenient and less error-prone when the same resource is always used.

Figure 5-1 shows optional methods of assigning LUNOs. Notice that a LUNO that is assigned using an SCI command may be either global or job-local, depending upon which command is used. A global LUNO is available to any task in the system; a job-local LUNO is only available to tasks of the job. Unless the LUNO must be available outside the current job, the job-local LUNO should be used. When a LUNO is assigned by an SVC, it may be a task-local LUNO, available only to the task. It also may be job-local or global. Once assigned, a LUNO must be opened by a task in order for that task to use the LUNO. Only one task is allowed to open and use one of these LUNOs at a given time.

Job-local-shared LUNOs (shared LUNOs) are job-local LUNOs that can be used by more than one task within a given job. The LUNO must be opened by any task that uses it. The access privileges of the LUNO are compared to those requested in the Open operation. If the Open operation requests greater access privileges and it does not conflict with the access privileges of other LUNOs that are assigned and opened to the resource, the privilege level of the LUNO is changed to the greater value. The access privileges of a LUNO in order of increasing value are read only, shared, exclusive write, and exclusive all. If the requested access privilege is less than or equal to the present value, the privilege level of the LUNO is not changed. Thus, all tasks that use a shared LUNO have the same access privileges to the resource regardless of how they opened it.

A count of the number of successful Open operations is kept. The same number of Close operations must be performed before the LUNO can be released. If a Close operation is not performed, the LUNO is not released until the job terminates.

The use of shared LUNOs tends to reduce the total number of LUNOs required in the system. This type of LUNO is not recommended for sequential files because there is no defined method of positioning the file; that is, the task has no control of which record is read or written.



2279463

Figure 5-1. Overall I/O Operation

5.2.1 Using Logical Names

The logical name is a useful way to represent a long pathname. It is essential for logically concatenated files and multifile sets. Logical names can be local to the job in which they are created or global to the system.

The Name Management SVC (opcode >43) includes a number of sub-opcodes related to logical names and synonyms. The sub-opcodes request the pathname that corresponds to a logical name and perform other functions for the system and for the task. The SVC performs similar functions for synonyms. However, since synonyms are SCI variables, they are normally assigned, modified, and deleted by SCI commands and subroutines. The sub-opcodes available to the user perform the following functions:

- Return the pathname and parameters for a logical name
- Create a logical name
- Delete a logical name
- Restore a name segment

A task that requires a new logical name performs a Set Name's Value (sub-opcode >02) operation, supplying the logical name, the value (pathname), and the parameters, if any. When a task needs the pathname or parameters of a logical name, it performs a Determine Name's Value (sub-opcode >00) operation. The task supplies the logical name for the operation. The Delete Name (sub-opcode >04) operation deletes a specified logical name. The Restore Name Segment (sub-opcode >0F) operation restores a name segment from disk.

The other sub-opcodes of the Name Management SVC perform operations used by the system. Systems programmers who need further details of these operations can find them in the *DNOS System Design Document.*

The supervisor call block for the Name Management SVC contains the following:

SVC >43 -- NAME MANAGEMENT ALIGN ON WORD BOUNDARY

DEC	HEX		
0	0	>43	<RETURN CODE>
2	2	SUB-OPCODE	FLAGS
4	4	ADDRESS OF NAME	
6	6	ADDRESS OF VALUE	
8	8	ADDRESS OF PARAMETER LIST	
10	A	SEGMENT ID/<PATHNAME FLAG>	
12	C	RESERVED	

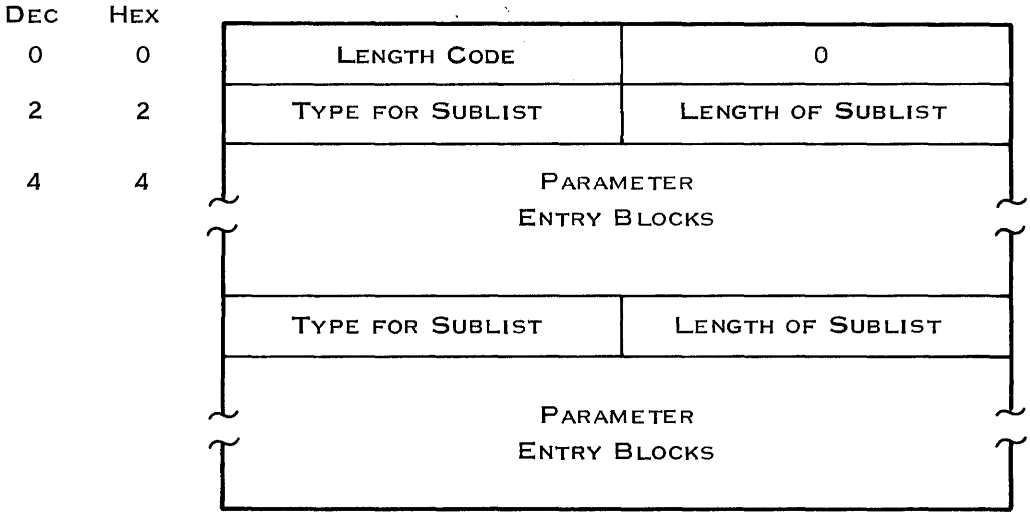
2279464

The call block contains the following:

Byte	Contents
0	Opcode, >43.
1	Return code. DNOS returns zero when the operation completes satisfactorily. When the operation completes in error, DNOS returns an error code.
2	Sub-opcode, as follows: >00 — Determine Name's Value. >02 — Set Name's Value. >04 — Delete Name. >0F — Restore Name Segment.
3	Flags. Bit 0 — Name type. Set as follows: 1 — Logical Name. 0 — Synonym. Bits 1-2 — Reserved. Bit 3 — Global Name. Bits 4-7 — Reserved.
4–5	Address of name. Address of a buffer that contains the name. The first byte of the buffer contains the length of the name.
6–7	Address of value. Address of a buffer that contains a pathname or other value for a logical name. The first byte of the buffer contains the length of the pathname.
8–9	Address of parameter list. Address of a buffer that contains a parameter list for a logical name. The first byte of the buffer contains the length of the list.
10–11	Pathname flag, used by the Determine Name's Value operation.
12–13	Reserved.

The task that requests a Set Name's Value operation for a logical name may supply parameters in a list. Also, if the name is a logical name that has a parameter list, the Determine Name's Value operation returns a parameter list.

These parameters can be set to create a file at a later time using these same parameters. Examples of operations that create files in this manner are the Create File operation or the Assign LUNO operation with autocreate option. The format of the parameter list is as follows:



2279465

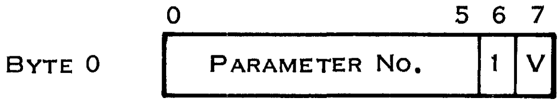
The parameter list contains the following:

Byte	Contents
0	Length code. The total length of the structure in bytes minus the length code byte.
1	Zero.

For each sublist:

Byte	Contents
2	Type for sublist. The type of the parameters in the sublist. Types of parameters are: 0 — System parameters. 1 — Spooler parameters. 2->7F — Reserved. >80->FF — User IPC parameters.
3	Length of sublist. The sum of the lengths of all parameter entry blocks in the sublist.
4-3 + n	Parameter entry blocks, one for each parameter. Formats of parameter entry blocks are described in subsequent paragraphs.

Three formats are defined for parameter entry blocks, any of which may be used for any type of parameter. The three formats are related to three parameter sizes. A parameter may be a single-bit binary value (a flag, for example). A parameter may be a value that can be stored in one byte. Or a parameter may occupy more than one byte. Each parameter format includes a parameter number and one or two bits that identify the format. The parameter entry block format for a single-bit value is:



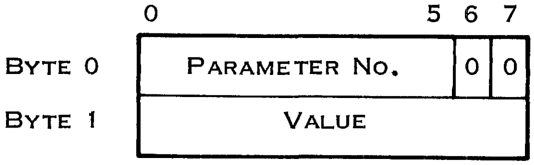
2279466

File Parameters

Parameter Name	Parameter Number
Job Temporary	05
Expandable	0D
Blank Suppression	0F
Forced Write	0E

Bit	Contents
0-5	Parameter number, 0 through 63. Parameter numbers need not be assigned or ordered in sequence but must be unique within the sublist.
6	1.
7	Value, 0 or 1.

The parameter entry block format for a one-byte parameter is:



2279467

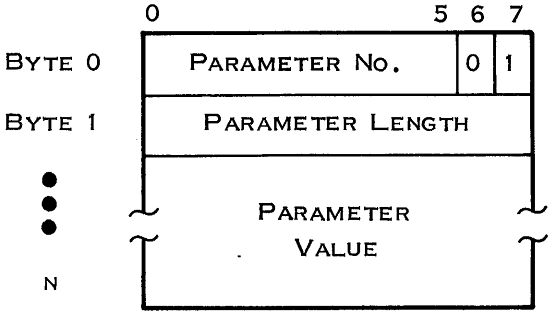
File Parameters

Parameter Name	Parameter Number
Maximum Number of Tasks	10
Maximum Number of Procedures	11
Maximum Number of Overlays	12
Job Access Level	03
File Type	04

The parameter entry block contains the following:

Byte	Contents
0	Parameter number byte: Bits 0-5 — Parameter number, 0 through 63. Parameter numbers need not be assigned or ordered in sequence but must be unique within the sublist. Bit 6 — 0. Bit 7 — 0.
1	Value. A numeric value, 0 through 255, or an ASCII character.

The parameter entry block format for a multibyte parameter is:



2279468

File Parameters

Parameter Name	Parameter Number
Initial Allocation	06
Physical Record Length	09
Default Physical Record Length	15
Secondary Allocation	07
Logical Record Length	08
Maximum Number of Directory Entries	14
Key Definition Block	16

The parameter entry block contains the following:

Byte	Contents
0	Parameter number byte: Bits 0-5 — Parameter number, 0 through 63. Parameter numbers need not be assigned or ordered in sequence but must be unique within the sublist. Bit 6 — 0. Bit 7 — 1.
1	Parameter length. The number of bytes required for the parameter value.
2-n	Parameter value. The numbers or characters of the parameter. If the parameter is a key definition block the parameter must be on a word boundary; that is, an even byte address.

The parameter list consists of one or more sublists. All parameters in a sublist are of the same type. Each parameter is defined in a parameter entry block, the format of which depends on the size of the parameter. Each parameter is identified by a parameter number in the range of 0 through 63. The parameters in a sublist must have unique parameter numbers. They may be numbered in any sequence, skipping numbers or not, as required.

5.2.1.1 Obtaining Logical Name Pathname and Parameters. The Determine Names's Value operation (sub-opcode >00) returns the pathname (or the first of a list of pathnames) and a parameter list, if any, and indicates whether or not the logical name has more pathnames. All fields of the supervisor call block apply to the operation.

When the flag (bit 0 of the flags byte) is set to zero, the operation expects the name to be a synonym and returns its value. You should call SCI subroutine S$MAPS to obtain the value of a synonym instead of requesting this operation. S$MAPS is discussed in the *DNOS Systems Programmer's Guide*.

The address of name field is the address of a buffer that contains a logical name. The first byte of the buffer must contain the number of bytes in the name.

The address of value field contains the address of a buffer large enough to hold the pathname expected. The first byte of the buffer contains the number of bytes in the buffer (not including the length byte). The operation returns the number of bytes in the pathname, followed by the characters of the pathname, in the buffer. When the logical name is not found, the operation returns an error code in byte 1 of the call block.

The address of parameter list field contains the address of a buffer large enough to contain the parameter list expected. The first byte of the buffer contains the number of bytes in the buffer (not including the length byte). The operation returns the parameter list in the format previously described, placing the list in the buffer. When no parameter list is found, the operation returns 0 in the first byte of the buffer.

The operation sets the pathname flag to 1 when the pathname returned by the operation is the first of a set of pathnames. The operation returns 0 in the field when the logical name represents only one pathname.

The following is an example of coding for a supervisor call block for a Determine Name's Value operation and for the required buffers:

```
        EVEN                  OBTAIN THE PATHNAME AND
DNMVAL  BYTE >43              PARAMETERS FOR LOGICAL NAME
DVER    BYTE 0                INFILE
        BYTE 0
        BYTE >80
        DATA LNAME
        DATA PATH
        DATA PARMS
MORE    DATA 0
        DATA 0
LNAME   BYTE 6
        TEXT 'INFILE'
PATH    BYTE 50
        BSS 50
PARMS   BYTE 100
        BSS 100
```

5.2.1.2 Creating a Logical Name. The Set Name's Value operation (sub-opcode >02) assigns a pathname and, optionally, a set of parameters to a logical name. A logical name segment must have been supplied for the current job.

The following fields of the supervisor call block apply:

- Opcode — >43
- Return code
- Sub-opcode — >02
- Flags
- Address of name
- Address of value
- Address of parameter list

When the global name flag in the flags byte is set to one for a logical name, the name is defined in the global name segment. To save this name for permanent use, issue the Snapshot Global Name Definition (SGND) command.

When the name type flag in the flags byte is set to zero, the operation expects the name to be a synonym and assigns its value. You should call SCI subroutine S$SETS to assign the value of a synonym instead of requesting this operation. S$SETS is discussed in the *DNOS Systems Programmer's Guide*.

The address of name field must contain the address of a buffer that contains the length of the name in the first byte and the characters of the logical name in succeeding bytes.

The address of value field must contain the address of a buffer that contains the pathname. The first byte in the buffer contains the length of the pathname. Successive bytes contain the characters of the pathname.

The address of parameter list field contains the address of a buffer or zero (when there are no parameters). The buffer contains the required parameters in the format previously described.

The following is an example of coding for a supervisor call block for a Set Name's Value operation and for the required buffers:

```
          EVEN                        CREATE LOGICAL NAME OUTFILE
CRLNAM    BYTE >43
CNER      BYTE 0
          BYTE >02
          BYTE >80
          DATA LNME
          DATA PTHNME
          DATA PARM
          DATA 0
          DATA 0
LNME      BYTE 7
          TEXT 'OUTFILE'
PTHNME    BYTE 11
          TEXT '.BLUE.FILE1'
PARM      BYTE 10
          BYTE 0
          BYTE >80
          BYTE 7
PARM1     BYTE >05
          BYTE 5
          TEXT 'LOCAL'
```

5.2.1.3 Deleting a Logical Name. The Delete Name operation (sub-opcode >04) deletes a logical name. The following fields of the supervisor call block apply:

- Opcode — >43
- Return code
- Sub-opcode — >04
- Flags
- Address of name

When the name type flag in the flags byte is set to zero, the operation expects the name to be a synonym, and attempts to delete the synonym. You should call SCI subroutine S$SETS to delete a synonym instead of requesting this operation.

The address of name field must contain the address of a buffer that contains the length of the name in the first byte and the characters of the logical name in succeeding bytes.

The following is an example of coding for a supervisor call block for a Delete Name operation using the name buffer of the previous example:

```
         EVEN                      DELETE LOGICAL NAME OUTFILE
DLN      BYTE >43
DNER     BYTE 0
         BYTE >04
         BYTE >80
         DATA LNAM
         DATA 0
         DATA 0
         DATA 0
         DATA 0
```

5.2.1.4 Restoring a Name Segment. The Restore Name Segment operation (sub-opcode >0F) restores a logical name segment from a disk file. The following fields of the supervisor call block apply:

- Opcode — >43
- Return Code
- Sub-opcode >0F
- Flags
- Address of name
- Segment ID

The only flag examined is the global flag. This operation can be performed only once. This is done by the system restart task.

The address of name field must contain the address of a buffer that contains the length of the name in the first byte and the characters of the logical name definition in succeeding bytes.

The address of value field must be zero.

The address of parameter list field must be zero.

If the user flag for global operation is set to 1, the global name operation will be done once, otherwise a user segment will be created.

The following is an example of coding for a supervisor call block for a Restore Name Segment operation and for the required buffers:

```
         EVEN                         RESTORE NAME SEGMENT
RESNAM   BYTE >43
RENS     BYTE 0
         BYTE >0F
         BYTE >00
         DATA LNME
         DATA 0
         DATA 0
         DATA 0
RID      DATA 0
LNME     BYTE 11
         TEXT '.DISK.FILE2'
```

5.2.2 Performing Utility Functions

Some of the sub-opcodes of the I/O Operations SVC (opcode >00) perform I/O utility functions that support device I/O, file I/O, and IPC. The Device I/O, File I/O, and Interprocess Communication sections in this manual describe these operations in detail. These I/O utility functions allow a program to:

- Create a file
- Delete a file
- Assign a LUNO to a file or device
- Release a LUNO
- Assign a new pathname to a file
- Verify a pathname
- Apply write protection to a file
- Apply delete protection to a file
- Remove protection from a file
- Add an alias to a directory
- Delete an alias in a directory
- Specify the write mode
- Create an IPC channel
- Delete an IPC channel

The utility operations require an extended supervisor call block. The following block applies to all utility functions except creating and deleting an IPC channel:

SVC >00 -- I/O Operations
(Utility sub-opcode)

Align on word boundary
Can be initiated as an event

Dec	Hex		
0	0	>00	<Return Code>
2	2	Sub-Opcode	LUNO
4	4	<System Flags>	User Flags
6	6	<Resource Type>	
8	8	Reserved	
10	A		
12	C	Key Def. Block Addr/Def., Phys. Rec. Size	
14	E	Reserved	
16	10	Utility Flags	
18	12	Defined Logical Record Length	
20	14	Defined Physical Record Length	
22	16	Pathname Address	
24	18	Parameter Address	
26	1A	Reserved	
28	1C	Initial File Allocation	
30	1E		
32	20	Secondary File Allocation	
34	22		

2279581

The call block contains the following:

Byte	Contents
0	Opcode, >00.
1	Return code. DNOS returns zero when the operation completes satisfactorily. When the operation completes in error, DNOS returns an error code.
2	Utility sub-opcode for desired operation: 90 — Create File. 91 — Assign LUNO. 92 — Delete File. 93 — Release LUNO. 95 — Assign New Pathname. 96 — Unprotect File. 97 — Write Protect File. 98 — Delete Protect File. 99 — Verify Pathname. 9A — Add Alias. 9B — Delete Alias. 9C — Define Write Mode. 9D — Create IPC Channel. 9E — Delete IPC Channel.
3	Logical unit number (LUNO).
4	<System flags>. Set by the system to indicate the status of the operation. Only the error flag (bit 1) is used for utility operations.
5	User flags. User sets these flags for utility operations. The flags for those operations that use this field are specified and described in the paragraph on each operation. Flag bits marked as not used must be set to 0 to avoid unpredictable results.
6–7	<Resource type>. Returned by the system for Assign LUNO operations.
8–11	[Reserved].
12–13	Key definition block address. When creating a key indexed file, place the address of the definition block in this field. Default physical record size. When creating a directory file, place the default physical record size in this field.
13	Number of tasks. When creating a program file, place the maximum number of tasks in this field.
14	Number of procedures. When creating a program file, place the maximum number of procedures in this field.

Byte	Contents
15	Number of overlays. When creating a program file, place the maximum number of overlays in this field.
16–17	Utility flags. User sets these flags for utility operations. The flags for each operation are specified and described in the paragraph on each operation. Set all flag bits that are marked as not used to zero to avoid unpredictable results.
18–19	Logical record length. Applies to create operations. Set to the number of bytes in the record or in the longest record (variable length records). A default value appropriate to the file type is used when this field contains zero.
20–21	Physical record length. Applies to create operations. Set to the number of bytes in the physical record. When this field contains any value less than twice the logical record length, the file is unblocked. A default value appropriate for the type of disk is used when this field contains zero.
22–23	Pathname address. Applies to all operations except Release LUNO. Set to the address of a field in memory that contains the following: Byte 0 — Length n of pathname in bytes Bytes 1-n — Pathname
24–25	Parameter address. See the specific sub-opcode description for usage.
26–27	Reserved.
28–31	Initial file allocation. Applies to a create operation. For an expandable file, set to the number of logical records to be allocated initially or to zero for the default value. For a fixed size file, set to the size of the total file, in logical records.
30–31	Directory entries. Applies to a Create File operation for a directory file. Set to the maximum number of directory entries.
32–35	Secondary file allocation. Applies to a create operation for an expandable file. Set to the number of logical records for subsequent allocations or to zero for the default value.

5.3 I/O OPERATIONS SVC

The I/O Operations SVC (opcode >00) is common to all I/O operations. A basic subset of sub-opcodes applies to resource-independent I/O. Other subsets apply to other types of I/O, as listed in Table 5-2. A basic supervisor call block also applies to resource-independent I/O. The extensions to and variations in the supervisor call block for other types of I/O are shown in the paragraphs that describe the types of I/O. The basic supervisor call block is as follows:

SVC >00 -- I/O OPERATIONS

ALIGN ON WORD BOUNDARY
CAN BE INITIATED AS AN EVENT

DEC	HEX		
0	0	>00	<RETURN CODE>
2	2	SUB-OPCODE	LUNO
4	4	<SYSTEM FLAGS>	USER FLAGS
6	6	DATA BUFFER ADDRESS	
8	8	READ CHARACTER COUNT	
10	A	WRITE CHARACTER COUNT/<ACTUAL READ COUNT>	

2279470

Table 5-2. Sub-Opcodes for I/O Operations SVC

Type of I/O	Sub-Opcodes (Hexadecimal)
Resource Independent	00-02, 05-07, 09-0F
Direct Disk	00, 03, 05, 08-0C, 0E-12
Sequential and Relative Record File	00-07, 09-0E, 10-12, 4A, 59, 5B
Key Indexed File	00, 01, 03, 05-07, 09, 0E, 40-4A, 50-52
911 VDT and 940 EVT	00-05, 09-0C, 0E, 15
Teleprinter Devices	00-05, 09-0D, 13, 15
Cassette Unit	00-07, 09-0F
Printer	00-04, 0B, 0D, 0E
Magnetic Tape	00-07, 09-0F
Card Reader	00, 01, 03, 04, 09, 0A
Interprocess Communication	00-05, 09-0D, 19-1C
Dummy Device	00-0F
I/O Utilities	90-93, 95-9E

The call block contains the following:

Byte	Contents
0	Opcode, >00.
1	Return code. DNOS returns zero when the operation completes satisfactorily. When the operation completes in error, DNOS returns an error code.
2	Sub-opcode for desired operation, as listed in Table 5-2. The basic I/O sub-opcodes are: 00 — Open. 01 — Close. 02 — Close, Write EOF. 03 — Open and Rewind. 04 — Close and Unload. 05 — Read Device Status. 06 — Forward Space. 07 — Backward Space. 09 — Read ASCII. 0A — Read Direct. 0B — Write ASCII. 0C — Write Direct. 0D — Write EOF. 0E — Rewind. 0F — Unload.
3	Logical unit number (LUNO).
4	<System flags> Set by the system to indicate the status of the operation. The flags that apply to each operation are specified and described in the paragraph on the operation.
5	User flags. Set to define the required operation. The flags that apply to each operation are specified and described in the paragraph on the operation. Flag bits marked as not used must be set to 0 to avoid unpredictable results.
6–7	Data buffer address. The address of the buffer for the operation; it must be on a word boundary.
8–9	Read character count. For a Read operation, the maximum number of characters that may be stored in the buffer.
10–11	Write character count. For a Write operation, the number of characters to be written. <Actual read count>. For a Read operation, set by the system to the number of characters stored in the buffer.

5.3.1 Suspending a Task During I/O

When I/O has been requested in the initiate mode, control returns to the task after the I/O has been initiated. The task can complete any processing that does not require the results of the I/O operation. When no further processing can be done, the task can issue an SVC to suspend itself until I/O is complete. DNOS supports two SVCs for this purpose. One suspends the calling task until a specified I/O operation has completed; the other suspends the calling task until any pending I/O operation initiated by the task completes.

5.3.1.1 Wait for I/O SVC. The Wait for I/O SVC (opcode >01) suspends the calling task until a specified I/O operation has completed. When the I/O has already completed, DNOS returns control to the calling task immediately.

The supervisor call block for the SVC is as follows:

SVC >01 -- WAIT FOR I/O   ALIGN ON WORD BOUNDARY

DEC	HEX		
0	0	>01	<RETURN CODE>
2	2	ADDRESS OF BYTE 2 OF I/O CALL BLOCK	

2279471

Byte	Contents
0	Opcode, >01.
1	Return code. DNOS returns zero when the operation completes satisfactorily. When the operation completes in error, DNOS returns an error code.
2-3	I/O address, byte 2 of the supervisor call block for the initiated I/O operation.

The following is an example of coding for a supervisor call block for a Wait for I/O SVC:

```
WFIO     DATA >0100          WAIT FOR COMPLETION OF I/O
         DATA IKIF+2         CALL BLOCK IKIF
```

5.3.1.2 Wait for Any I/O SVC. The Wait for Any I/O SVC (opcode >36) suspends the calling task until an I/O operation requested by the task completes. When the task resumes execution, it must test the busy flag (byte 4, bit 0) in the supervisor call block for each I/O operation to identify the completed operation. When all I/O has already completed, DNOS returns control to the calling task immediately.

The supervisor call block for the SVC is as follows:

SVC >36 -- WAIT FOR ANY I/O ALIGN ON WORD BOUNDARY

DEC	HEX		
0	0	> 36	<RETURN CODE>

2279472

The call block contains the following:

Byte	Contents
0	Opcode, >36.
1	Return code. DNOS returns zero when the operation completes satisfactorily. When the operation completes in error, DNOS returns an error code.

The following is an example of coding for a supervisor call block for a Wait for Any I/O SVC:

```
WMIO     DATA >3600                 WAIT FOR COMPLETION OF I/O
```

5.3.2 Forcing Termination of I/O

DNOS supports a supervisor call that forces termination of I/O to the device assigned to a specified LUNO. The SVC can abort I/O to the device from the calling task or from another task. Only privileged tasks may abort I/O from another task. The Abort I/O SVC (opcode >0F) aborts I/O to the device, optionally closing the device when it has been opened. If the device is busy, the SVC sets the error flag (byte 4, bit 1) in the supervisor call block for the aborted operation. The calling task is suspended during execution of the SVC. The medium involved in the I/O operation remains positioned as the aborted I/O operation leaves it; that is, a tape is not rewound or backspaced.

The supervisor call block for the SVC is as follows:

SVC > 0F -- Abort I/O   Privileged task (see text)

Dec	Hex		
0	0	>0F	<Return Code>
2	2	Flags	LUNO
4	4	Zero, or Address of TSB	
6	6	Zero, or Address of JSB	

2279473

The call block contains the following:

Byte **Contents**

0 Opcode, >0F.

1 Return code. DNOS returns zero when the operation completes satisfactorily. When the operation completes in error, DNOS returns an error code.

2 Flags.
Bit 0 — Do not close. Set as follows:
  1 — Do not close files and devices.
  0 — Close open files and devices.
Bits 1-7 — Reserved.

3 LUNO assigned to the device to which I/O is to be aborted.

4-5 Zero, when calling task I/O is to be aborted. Address of Task Status Block (TSB) of job for which I/O is to be aborted.

6-7 Zero, when calling task I/O is to be aborted. Address of Job Status Block (JSB) of task for which I/O is to be aborted.

The following is an example of coding for a supervisor call block for an Abort I/O SVC:

```
AIO     DATA >0F00          ABORT I/O TO LUNO >3E FROM
        DATA >003E          THIS TASK, CLOSING FILES
        DATA 0
        DATA 0
```

6

Device I/O

6.1 INTRODUCTION

This section describes the utility operations and the I/O operations for device I/O. The descriptions of the utility operations apply to all devices. Descriptions of the applicable I/O operations for each device follow. These descriptions are organized by device.

6.2 DEVICE UTILITY OPERATIONS

Several utility operations are required to support device I/O. A device may be specified by either a device name or by a logical name. A logical name promotes documentation of the program; it is local to a job; and it may associate parameters with the device. Logical name operations are described in the Input/Output Operations section in this manual. Utility operations include assignment and deletion of LUNOs by the I/O utility.

The I/O utility functions for device I/O are:

- Assign a LUNO
- Release a LUNO
- Verify a device name

The following extended supervisor call block applies to utility functions.

SVC >00 -- I O OPERATIONS
(UTILITY SUB-OPCODE)

ALIGN ON WORD BOUNDARY
CAN BE INITIATED AS AN
EVENT

DEC	HEX		
0	0	>00	<RETURN CODE>
2	2	SUB-OPCODE	LUNO
4	4	<SYSTEM FLAGS>	USER FLAGS
6	6	<RESOURCE TYPE>	
8	8	RESERVED	
10	A		
12	C	KEY DEF. BLOCK ADDR/DEF. PHYS. REC. SIZE	
14	E	RESERVED	
16	10	UTILITY FLAGS	
18	12	DEFINED LOGICAL RECORD LENGTH	
20	14	DEFINED PHYSICAL RECORD LENGTH	
22	16	PATHNAME ADDRESS	
24	18	PARAMETER ADDRESS	
26	1A	RESERVED	
28	1C	INITIAL FILE ALLOCATION	
30	1E		
32	20	SECONDARY FILE ALLOCATION	
34	22		

2279581

6.2.1 Assigning LUNOs

To assign a LUNO, a program executes an I/O Operations SVC with sub-opcode >91. The following fields of the utility supervisor call block apply:

- SVC code — 0
- Return code
- Utility sub-opcode — >91
- Logical unit number (LUNO)
- <Resource type>
- Utility flags
- Pathname address

The system returns the resource type in bytes 6 and 7 of the call block. Byte 6 indicates the device type and byte 7 indicates the resource type.

The device types found in byte 6 are as follows:

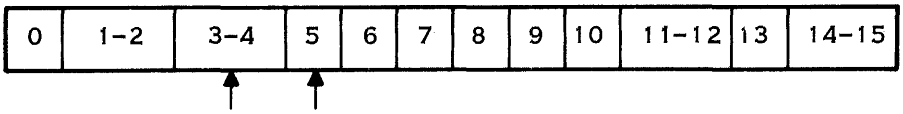
>00	Dummy device
>01	Special device
>02	743 KSR
>03	733 ASR
>04	733 cassette drive
>06	Single-sided diskette drive
>07	Disk drive
>08	Magnetic tape drive
>09	Teleprinter device (TPD)
>0A	911 VDT
>0B	Serial printer
>0C	Parallel printer
>0D	Four-channel communication controller (FCCC)
>0E	Communication interface module (CIM)
>0F	Industrial device
>10	Card reader
>11	940 VDT
>12	931 VDT
>14	Bit-oriented/character-oriented asynchronous interface module (BCAIM)
>15	Virtual terminal

The resource types found in byte 7 are as follows:

>01	File
>02	Device
>04	Channel
>08	Remote

The value in byte 7 is formed by DNOS by using one or more of the values listed above. Some of the values are combinations of these values. For example, >06 is a channel emulating a device. If the value in bytes 6 and 7 is 0A06, the indicated resource type is a channel emulating a 911 VDT.

The following utility flags apply:



2279474

Bits 3-4 — Scope of LUNO flag. Set as follows:

00 — Task-local LUNO.
01 — Job-local LUNO.
10 — Global LUNO.
11 — Job-local-shared LUNO.

Bit 5 — Generate LUNO flag. Set as follows:

1 — Assign the next available LUNO and return it in byte 3.
0 — Assign the LUNO specified in byte 3.

Set all other utility flags to zero.

A logical unit number (LUNO) must be assigned to an I/O resource to identify the resource for an I/O operation. The scope of a global LUNO is not limited to a single job or task. The LUNO applies in all jobs and tasks executing while it remains assigned. The scope of a job-local LUNO is limited to the tasks in the job. A job-local LUNO is assigned by one of the tasks in the job or by an SCI command. The scope of a task-local LUNO is limited to the task that assigns the LUNO. A task-local LUNO is assigned by a task.

Job-local-shared LUNOs (shared LUNOs) are job-local LUNOs that can be used by more than one task within a given job. Each task that uses the LUNO must open it. The access privileges of the LUNO are compared to those requested in the Open operation. If the Open operation requests greater access privileges and it does not conflict with the access privileges of other LUNOs that are assigned and opened to the resource, the privilege level of the LUNO is changed to the greater value. The access privileges of a LUNO in order of increasing value are:

- Read only
- Shared
- Exclusive write
- Exclusive all

If the requested access privilege is less than or equal to the present value, the privilege level of the LUNO is not changed. Thus, all tasks that use a shared LUNO have the same access privileges to the resource regardless of how they opened it.

A count of the number of successful Open operations is kept. The same number of Close operations must be performed before the LUNO can be released. If a Close operation is not performed, the LUNO is not released until the job terminates.

The use of shared LUNOs tends to reduce the total number of LUNOs required in the system. This type of LUNO is not recommended for sequential files because there is no defined method of positioning the file; that is, the task has no control over which record is read or written.

The Assign LUNO operation may assign the next available LUNO or a LUNO specified in the LUNO field. When the generate LUNO flag is set to one, the system assigns the next available LUNO and returns the number in the LUNO field. When the flag is set to zero, the system considers the contents of the LUNO field of the supervisor call block to be the desired LUNO.

The pathname address is the address of an area of memory that contains the pathname (device name) of a resource to be assigned to the LUNO. The first byte of the pathname area contains four, the number of characters in the device name. Subsequent bytes contain the ASCII characters of the device name.

The following is an example of the source code for a supervisor call block and a device name block to assign a LUNO to a device:

```
ALUNO    DATA 0                  ASSIGN NEXT AVAILABLE JOB LOCAL
         BYTE >91                LUNO TO LINE PRINTER
         BYTE 0
         DATA 0,0
         DATA 0,0
         DATA 0,0
         BYTE >0C,0              UTILITY FLAGS
         DATA 0,0
         DATA DNME
         DATA 0,0
         DATA 0,0
         DATA 0,0
DNME     BYTE 4                  DEVICE NAME LENGTH
         TEXT 'LP02'
```

6.2.2 Releasing LUNOs

To release a LUNO, a program executes an I/O Operations SVC with sub-opcode >93. The following fields of the utility supervisor call block apply:

- SVC code — 0
- Return code
- Utility sub-opcode — >93
- LUNO
- Utility flags

The following utility flags apply:

0	1–2	3–4	5	6	7	8	9	10	11–12	13	14–15
		↑									

2279475

Bits 3-4 — Scope of LUNO. Set as follows:
00 — Task-local LUNO
01 — Job-local LUNO
10 — Global LUNO
11 — Job-local-shared LUNO

Set all other utility flags to zero.

A Release LUNO operation does not release a LUNO that has a different scope from that specified by the scope of LUNO flag. For example, if global LUNO >23, job-local LUNO >23, and task-local LUNO >23 were all assigned, and a Release LUNO operation for task-local LUNO >23 were performed, the global and job-local LUNOs would remain assigned.

The following is an example of the source code for a supervisor call block to release a LUNO:

```
RLUNO     DATA 0                     RELEASE GLOBAL LUNO >23.
          BYTE >93
          BYTE >23
          DATA 0,0
          DATA 0,0
          DATA 0,0
          BYTE >10,0                 UTILITY FLAGS
          DATA 0,0
          DATA 0
          DATA 0,0
          DATA 0,0
          DATA 0,0
```

6.2.3 Verifying Device Names

To verify a device name, a program executes an I/O Operations SVC with sub-opcode >99. The following fields of the utility supervisor call block apply:

- SVC code — 0
- Return code
- Utility sub-opcode — >99
- Resource type
- Utility flags
- Pathname address

DNOS returns the resource type in bytes 6 and 7 of the supervisor call block, just as it does for the Assign LUNO operation.

The utility flags are set to zero to verify a device name.

The Verify Device Name operation performs a syntax check on the device name.

The pathname address is the address of an area of memory that contains the device name to be verified. The byte at the pathname address contains four, the number of characters in the device name. Subsequent bytes contain the ASCII characters of the device name.

The following is an example of the source code for a supervisor call block to verify a device name:

```
VFY       DATA 0                 VERIFY DEVICE NAME LP01
          BYTE >99,0
          DATA 0,0
          DATA 0,0
          DATA 0,0
TYPE      BYTE 0,0               UTILITY FLAGS
          DATA 0,0
          DATA DEVNA
          DATA 0,0
          DATA 0,0
          DATA 0,0
DEVNA     BYTE 4
          TEXT 'LP01'
```

6.3 VDT I/O

DNOS supports both resource-independent and resource-specific I/O for video display terminal (VDT) I/O. Resource-independent I/O for the VDT includes operations that are analogous to sequential file operations. A record is displayed on the bottom line of the screen as it is entered by the user or written by the program. Previously displayed lines move upward, and the top line disappears from the display. Specifically, resource-independent I/O to a terminal implies the following conditions.

An Open operation positions the cursor at column 0 of the bottom row of the screen.

The Write operation default conditions are:

- Output is displayed at the current cursor position.
- Characters are displayed at low intensity.
- A carriage return (>0D) positions the cursor at column 0 of the current row.
- A line feed (>0A) or form feed (>0C) moves the current line and the lines above it up one line and positions the cursor at column 0 of the current line.

The Read operation default conditions are:

- Characters entered are displayed at the current cursor position.
- Characters entered are displayed at high intensity.

Resource-specific I/O for VDTs include operations that apply only to a video display device. These operations give a program control over cursor position (which implies display position for input and output), audible tone, function of special keys, and intensity of display. Except for the Read Device Status operation, the device must be opened using sub-opcode >00 or >03 prior to any I/O operation.

The following I/O call block for VDT I/O operations is the basic block used for all I/O operations. If an extension to this block is necessary for a particular operation, it is indicated in the operation description.

SVC >00 -- I/O OPERATIONS

ALIGN ON WORD BOUNDARY
CAN BE INITIATED AS AN EVENT

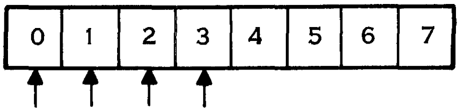
DEC	HEX		
0	0	>00	<RETURN CODE>
2	2	SUB-OPCODE	LUNO
4	4	<SYSTEM FLAGS>	USER FLAGS
6	6	DATA BUFFER ADDRESS	
8	8	READ CHARACTER COUNT	
10	A	WRITE CHARACTER COUNT/<ACTUAL READ COUNT>	

2279470

The subset of sub-opcodes for the VDT applies to both resource-independent and resource-specific I/O, as follows:

00	Open
01	Close
02	Close, Write EOF
03	Open and Rewind
04	Close and Unload
05	Read Device Status
09	Read ASCII
0A	Read Direct
0B	Write ASCII
0C	Write Direct
0D	Supported by 931/940
0E	Rewind
12	Supported by 931/940

The system flags (byte 4) in the supervisor call block apply to all VDT I/O. These flags are:



2279476

Bit 0 — Busy flag. Set by system as follows:
1 — Busy.
0 — Operation completed.

Bit 1 — Error flag. Set by system as follows:
1 — Error.
0 — No error.

Bit 2 — End-of-file flag. Set by system as follows:
1 — ENTER key terminated the operation.
0 — Operation terminated without the ENTER key being pressed.

Bit 3 — Event key flag. Set by system as follows:
1 — An event key terminated the operation.
0 — Operation terminated without an event key being pressed.

The user flags (byte 5) in the supervisor call block apply to all VDT I/O. However, significance of these flags differs for various operations. The flags that apply to each operation are described in the detailed description of each operation.

The character set applicable to the VDT is the ASCII character set or the JISCII character set appropriate to the country code for which the system was generated.

6.3.1 Key Categories

The system interprets each key on the keyboard in one or more of the following categories:

- Data
- Hold
- Event
- System edit
- Task edit

6.3.1.1 Data Keys. The data keys return the codes of printable characters to the buffer specified in the call block. The category includes the keys that return ASCII codes >20 through >7E.

6.3.1.2 Hold Key. The hold key suspends output to the terminal. Operation may be resumed by pressing any other key except RETURN, exclamation point (!), or CONTROL X. The hold key is the blank orange key on the 911 and 931, and the PREV FORM key on the 940.

When the RETURN key is pressed following the hold key, any write operation in progress is aborted.

When the exclamation point key is pressed following the hold key, the system activates SCI and the output continues. Contention between the interrupted output and SCI for the use of the terminal may cause unpredictable results.

When CONTROL X is pressed following the hold key, a hard break results. The hard break terminates a current task and activates SCI. When end action is specified by the task, it is performed prior to terminating the task and activating SCI. The hard break should be used to abort tasks when appropriate. It should be used with care because it aborts pending I/O requests.

The system selects the task to be terminated by the hard break from among the tasks of the current job. The following rules apply in the listed order of priority:

1. When only the System Command Interpreter (SCI) task is active, SCI is terminated.

2. Any other foreground task is terminated.

3. Any other background task is terminated.

4. When other tasks are active along with SCI, SCI is terminated last.

There may be times when the hard break terminates a task other than the one intended. In that case, press the hold key and CONTROL X again to terminate the intended task.

The effect of a subsequent hard break depends upon the timing. When the task has not yet taken end action in response to the first hard break, the subsequent hard break terminates the next task in the order of priority. When the task is executing the end action routine but has not completed end action, the task is aborted as if end action had not been provided. When end action has completed, the subsequent hard break causes the task to take end action again.

The end action routine should execute a Get End Action Status SVC to obtain the error code and identify the cause of the termination. When the task error code returned by the SVC is >10, a hard break has occurred. The task may process the hard break and resume execution after executing a Reset End Action Status SVC. The task must place the WP, PC, and ST values returned by the Get End Action Status SVC in R13, R14, and R15 and execute an RTWP instruction to resume execution.

6.3.1.3 Event Keys. Activating the event key mode enables use of event keys as task programmable function keys. The event key mode is activated by performing an Open operation with the event key mode bit (bit 7 of byte 5 of the call block) set to one. Event characters may be accessed by a Remote Get Event Character operation without opening the LUNO assigned to the VDT.

When an event key is pressed, the corresponding character code is stored in the event character buffer. When an input operation is being performed, the operation terminates with the event key bit (bit 3 of byte 4 of the call block) set to one.

When no input operation is being performed and an event key is pressed, the next input operation is immediately terminated with the event key bit of the call block set to one.

The events keys are identified in Table 6-2.

The task decodes the event character and performs the desired function. When the input operation that terminates with the event key bit set to one uses the extended call block (resource-specific I/O), the event character is returned in the event character field. The event character can be obtained without performing a read operation (and without opening the LUNO) by performing a Remote Get Event Character operation. When the input operation uses the basic I/O call block (resource-independent I/O), the event character is not returned.

NOTE

> Any task may access the event character buffer by performing I/O to any LUNO assigned to the terminal. The first access is the only access that returns the correct character. Tasks that perform I/O to a terminal to which SCI is performing I/O must avoid accessing event characters. Either the task or SCI may fail to perform the intended function.

6.3.1.4 System and Task Edit Keys. System edit keys are cursor and display control keys that are implemented by the system. Task edit keys are cursor and display control keys that are task functions. Five of the keys are both system and task edit keys, for which the system performs functions. The task may perform additional functions.

Task edit functions do not apply to resource-independent I/O. The carriage control bit in the extended user flags field must be set to one to enable task edit functions. When task edit functions are enabled, pressing a task edit key during an input operation terminates the operation. The device service routine (DSR) returns the character code of the task edit character in the event byte field of the extended call block.

The following paragraphs describe the edit keys. For system edit keys, the paragraphs describe the functions performed when the keys are pressed. For task edit keys, the paragraphs state the code that the DSR returns to the task when the keys are pressed. For keys that are both, the paragraphs describe the functions and state the codes. If a key is both and the task edit flag is not set, the function is performed, but the I/O is not terminated. System edit keys are listed with a letter S in the type column of Table 6-2; task edit keys are listed with a letter T in the type column.

Edit keys are related to fields of data on the screen, which are read in an I/O operation. The field begins at the cursor position and consists of the number of characters to be read by the current read operation. When an edit key is pressed between read operations, it is effective at the beginning of the next read operation.

The following paragraphs discuss the edit keys on a 911 VDT; equivalent keys on other terminals perform these functions. See the ASCII Device I/O Operations appendix to this manual for the equivalent keys on your terminal.

ERASE FIELD Key. This system edit key positions the cursor at the beginning of the field and fills the field with the fill character.

Left Arrow Key. This system edit key moves the cursor to the left unless the cursor is at the first character position of the field. The warning beep bit of the extended user flags field controls an audible warning if the cursor is not moved.

Right Arrow Key. This system edit key moves the cursor to the right one character position. When the cursor moves beyond the end of the field, and the remain in field on field full flag is not set, the DSR terminates the operation and returns the last character entered. The warning beep bit of the extended user flags field controls an audible warning if the cursor moves beyond the end of the field.

INS CHAR Key. This system edit key sets the input mode to insert characters. In the insert mode, entering a character moves the cursor and the characters to its right one character position to the right and displays the entered character in the position vacated by the cursor. Pressing the key causes a warning beep when the cursor is positioned at a fill character, if the fill character is not a blank, and the insert mode is not enabled. The warning beep bit of the extended user flags field controls an audible warning if the field has been filled. The insert mode remains effective until a key that is not a data key is pressed.

DEL CHAR Key. This system edit key deletes the character at the cursor position, moves the characters in the field to the right of the cursor one position to the left, and fills the left character position of the field with the fill character. The warning beep bit of the extended user flags field controls an audible warning if there is no character to be deleted.

Blank Gray Key. This task edit key returns the character code >8F in the event byte field.

Up Arrow Key. This task edit key returns the character code >95 in the event byte field.

ENTER Key. This task edit key returns the character code >93 in the event field byte. The DSR interprets this code as end-of-file (EOF) and sets bit 2 of byte 4 of the call block.

Left FIELD Key. This task edit key returns the character code >94 in the event byte field.

Right FIELD Key. This task edit key returns the character code >87 in the event byte field.

Down Arrow Key. This task edit key returns the character code >8A in the event byte field.

ERASE INPUT Key. This system and task edit key positions the cursor at the beginning of the field, fills the field with the fill character, and returns the character code >8E in the event byte field.

RETURN Key. This system and task edit key returns the cursor to column 1 of the current line and returns the character code >8D in the event byte field. The device service routine (DSR) interprets the code as end-of-record (EOR).

HOME Key. This system and task edit key returns the cursor to the beginning of the field and returns the character code >8C in the event byte field.

SKIP Key. This system and task edit key fills the field from the cursor position to the end of the field with fill characters without moving the cursor and returns the character code >8B in the event byte field.

TAB Key. This system and task edit key accepts all characters entered in the current field without moving the cursor, and returns the character code >89 in the event byte field.

6.3.2 VDT Resource-Independent I/O

The operations appropriate for the VDT are described in subsequent paragraphs. The following sub-opcodes, which do not apply to the VDT, produce the indicated results:

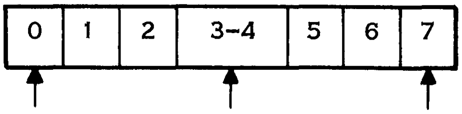
06	Ignored
07	Ignored
08	Error
0D	Ignored for 911
0F	Ignored

6.3.2.1 Open. Sub-opcode >00 specifies an Open operation. The Open operation is required for a VDT. However, DNOS does not validate the Open operation; that is, it does not detect a possible conflict with I/O to the same device by another task. An Open operation is not required prior to performing a Read Device Status operation.

The following fields of the basic supervisor call block apply to an Open operation:

- SVC code — 0
- Return code
- Sub-opcode — >00
- Logical unit number (LUNO)
- User flags
- Data buffer address
- Read character count

The following user flags apply to an Open operation:



2279478

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

Bits 3–4 — Access privilege flag. Set as follows:
00 — Exclusive write.
01 — Exclusive all.
10 — Shared.
11 — Read only.

Bit 7 — Event key mode flag. Set as follows:
1 — Enable event key mode.
0 — Disable event key mode.

The logical unit number (LUNO) field contains the LUNO assigned to the VDT to be opened.

The Open operation returns the device type code in the data buffer address field (bytes 6 and 7) of the supervisor call block. The device type code for the VDT is 5.

When the calling task places zero in the read character count field (bytes 8 and 9) of the supervisor call block, the Open operation returns the default logical record length for the device. The default logical record length for a VDT is >56.

A VDT must be opened with the event key mode flag set to one if event keys are to be used as task programmable function keys.

To access an event key character, perform a Remote Get Event Character operation.

The following is an example of the source code for a supervisor call block to open a VDT:

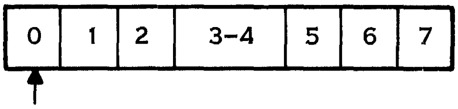
```
OVDT     DATA 0                  OPEN VDT ASSIGNED TO LUNO >24.
         BYTE 0,>24
         DATA 0
TYPE     DATA 0
DLRL     DATA 0
         DATA 0
```

6.3.2.2 Close. Sub-opcode >01 specifies a Close operation. The Close operation ends I/O to a LUNO from the calling task. The LUNO remains assigned to the device, and may be opened again for additional I/O operations. When a task terminates, DNOS closes all LUNOs that have been opened by the task.

The following fields of the basic supervisor call block apply to a Close operation:

- SVC code — 0
- Return code
- Sub-opcode — >01
- Logical unit number (LUNO)
- User flags

The following user flag applies to a Close operation:



2279479

Bit 0 — Initiate flag. Set as follows:

1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

The logical unit number (LUNO) field contains the LUNO to be closed.

The following is an example of the source code for a supervisor call block to close a VDT:

```
CVDT      DATA 0                      CLOSE VDT ASSIGNED TO LUNO >24.
          BYTE 1,>24
          DATA 0
          DATA 0
          DATA 0
          DATA 0
```

6.3.2.3 Close, Write EOF. The Close, Write EOF operation, sub-opcode >02, is identical to the Close operation.

6.3.2.4 Open and Rewind. The Open and Rewind operation, sub-opcode >03, is an Open operation followed by a Rewind operation. For the VDT, the Rewind operation consists of clearing the screen. Any previously entered characters that remain in the buffer are ignored.

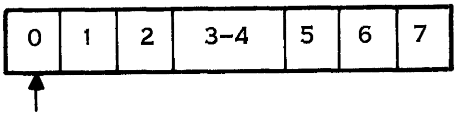
6.3.2.5 Close and Unload. The Close and Unload operation, sub-opcode >04, is identical to the Close operation.

6.3.2.6 Read Device Status. Sub-opcode >05 specifies a Read Device Status operation. The Read Device Status operation returns device status information in a buffer.

The following fields of the basic supervisor call block apply to a Read Device Status operation:

- SVC code — 0
- Return code
- Sub-opcode — >05
- Logical unit number (LUNO)
- User flags
- Data buffer address
- Read character count
- <Actual read count>

The following user flag applies to a Read Device Status operation:



2279480

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

The logical unit number (LUNO) field contains the LUNO assigned to the device for which status information is returned.

The data buffer address is the address of the buffer into which DNOS places the status information.

The read character count is the length of the buffer.

DNOS returns the number of characters stored in the buffer in the actual read count field. If the buffer is 4 characters, bytes 0 through 3 are returned. If the buffer is 18 characters, all bytes are returned.

The contents of the data buffer after a Read Device Status operation has returned the status of a VDT are:

Byte	**Contents**
0	Maximum row address.
1	Maximum column address.
2-3	Number of characters currently stored in the input character queue.
4	Device Service Routine (DSR) type.*
5	Channel number.*
6-7	Communications Register Unit (CRU) address.
8-9	Auto Call Unit (ACU) CRU address.*
10	Interface Service Routine (ISR) type.*
11	Line Control*
12–13	Opcode 15, Edit flag 1.

* For more information concerning these items, see the resource specific information for the appropriate device.

Byte	Contents
14–15	Opcode 15, Edit flag 2.
16–17	Reserved.

DNOS maintains an input character queue that stores characters input while the system is processing a previously-entered character or command. Bytes 2 and 3 contain the number of characters in the queue. The maximum size of this queue is specified when the system is generated.

The following is an example of the source code for a supervisor call block for a Read Device Status operation and code for the read buffer:

```
RDSVDT  DATA 0          READ STATUS OF VDT ASSIGNED TO
        BYTE 5,>32      LUNO >32.
        DATA 0
        DATA MRA
        DATA 18
        DATA 0
MRA     BSS 1           DEVICE STATUS BUFFER
MCA     BSS 1
CIQ     BSS 16
```

6.3.2.7 Read ASCII. Sub-opcode >09 specifies a Read ASCII operation. The Read ASCII operation reads a record from the keyboard and stores the characters in the specified buffer, two characters per word.

The following fields of the basic supervisor call block apply to a Read ASCII operation:

- SVC code — 0
- Return code
- Sub-opcode — >09
- Logical unit number (LUNO)
- <System flags>
- User flags
- Data buffer address
- Read character count
- <Actual read count>

Byte	Contents
14–15	Opcode 15, Edit flag 2.
16–17	Reserved.

DNOS maintains an input character queue that stores characters input while the system is processing a previously-entered character or command. Bytes 2 and 3 contain the number of characters in the queue. The maximum size of this queue is specified when the system is generated.

The following is an example of the source code for a supervisor call block for a Read Device Status operation and code for the read buffer:

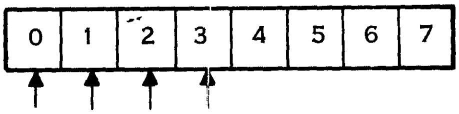
```
RDSVDT  DATA 0                 READ STATUS OF VDT ASSIGNED TO
        BYTE 5,>32             LUNO >32.
        DATA 0
        DATA MRA
        DATA 18
        DATA 0
MRA     BSS 1                  DEVICE STATUS BUFFER
MCA     BSS 1
CIQ     BSS 16
```

6.3.2.7 Read ASCII. Sub-opcode >09 specifies a Read ASCII operation. The Read ASCII operation reads a record from the keyboard and stores the characters in the specified buffer, two characters per word.

The following fields of the basic supervisor call block apply to a Read ASCII operation:

- SVC code — 0
- Return code
- Sub-opcode — >09
- Logical unit number (LUNO)
- <System flags>
- User flags
- Data buffer address
- Read character count
- <Actual read count>

he following system flags apply to a Read ASCII operation:



2279481

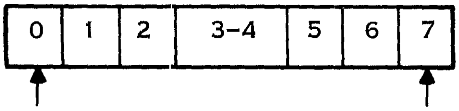
Bit 0 — Busy flag. Set by system as follows:
1 — Busy.
0 — Operation completed.

Bit 1 — Error flag. Set by system as follows:
1 — Error.
0 — No error.

Bit 2 — End-of-file. Set by system as follows:
1 — ENTER key terminated the operation.
0 — Operation terminated without the ENTER key being pressed.

Bit 3 — Event key flag. Set by system as follows:
1 — An event key terminated the operation.
0 — Operation terminated without an event key being pressed.

The following user flags apply to a Read ASCII operation:



2279482

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

Bit 7 — Blank adjustment flag. Set as follows:
1 — Read with blank adjustment.
0 — Read without blank adjustment.

The logical unit number (LUNO) field contains the LUNO assigned to the VDT from which a record is to be read.

The data buffer address is the address of the buffer into which DNOS places the record.

The read character count is the length of the buffer.

DNOS returns the number of characters stored in the buffer in the actual read count field.

The Read ASCII operation recognizes the characters listed in Appendix B for the VDT. The operation stores the characters, packed one per byte. When the country code in effect is not >0200 (Japan), the most significant bit is set to zero. When the country code is >0200, the eight-bit JISCII code is stored. The operaton continues until the RETURN key is pressed, the buffer is full, an event key is pressed (if the VDT is in the event key mode), or a task edit key is pressed (if task edit is set).

When the ENTER key is pressed, the system sets the EOF flag in the system flags byte and terminates the operation.

Characters entered between Read operations are stored in a queue and read by the next Read operation. The maximum size of the queue is specified during system generation. When the queue has been filled, the audible tone sounds as each additional character is entered and the additional characters are ignored.

Errors can be corrected by pressing the left arrow key to backspace the cursor to the character in error. Entering the correct character replaces the incorrect character. Only characters being corrected need to be reentered.

When the RETURN key is pressed, the number of characters entered is stored in the actual read count field and the operation terminates.

When the VDT is opened in the event key mode and an event key is pressed, the system sets the event key flag in the system flags byte and terminates the operation. The event character may be accessed by performing a Remote Get Event Character operation.

When an event key is pressed between Read operations, the next Read operation performed after the pressing of the event key terminates with the event key flag set.

When blank adjustment is specified for variable length records, blanks are stored in the buffer to fill the record. That is, when the record length is less than the buffer size, the device service routine (DSR) supplies blanks (>20) to fill the buffer. The character count returned in bytes 10 and 11 includes the blanks supplied by the DSR.

The following is an example of the source code for a supervisor call block for a Read ASCII operation and code for the read buffer:

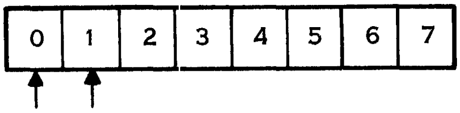
```
RDVDT    DATA 0              READ RECORD FROM VDT ASSIGNED TO
         BYTE 9,>2B          LUNO >2B IN THE INITIATE I/O MODE
         BYTE 0,>80
         DATA RBUFF
         DATA 80
         DATA 0
RBUFF    BSS 80              READ BUFFER
```

6.3.2.8 Write ASCII. Sub-opcode >0B specifies a Write ASCII operation. The Write ASCII operation transfers a record from the specified buffer to the screen of the VDT. DNOS supports a write with reply option, which is effectively a Write operation followed by a Read ASCII operation.

The following fields of the basic supervisor call block apply to a Write ASCII operation:

- SVC code — 0
- Return code
- Sub-opcode — >0B
- LUNO
- <System flags>
- User flags
- Data buffer address
- Write character count
- Reply block address (write with reply option)

The following system flags apply to a Write ASCII operation:

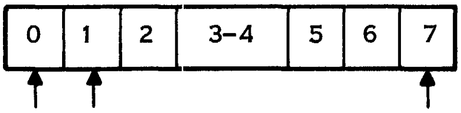


2279483

Bit 0 — Busy flag. Set by system as follows:
1 — Busy.
0 — Operation completed.

Bit 1 — Error flag. Set by system as follows:
1 — Error.
0 — No error.

The following user flags apply to a Write ASCII operation:



2279484

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

Bit 1 — Reply flag. Set as follows:
1 — Write operation followed by a Read operation.
0 — All other operations.

Bit 7 — Blank adjustment flag. Set as follows:
1 — Write with blank adjustment.
0 — Write without blank adjustment.

The logical unit number (LUNO) field contains the LUNO assigned to the VDT to which a record is to be written.

The data buffer address is the address of the buffer which contains the record to be displayed.

The write character count is the number of characters to be displayed on the screen of the VDT.

The Write ASCII operation displays a record on the screen of a VDT. The record consists of ASCII characters or JISCII characters, as specified by the country code.

The Write ASCII operation allows repeat character compression in the data to be displayed. Repeat character compression represents a string of identical characters (for example, underlines) as six hexadecimal digits. The first two digits are the hexadecimal representation of the ASCII code of the character to be repeated. The next two digits are >7F and the last two digits are the hexadecimal number of identical characters. For example, >20, >7F, >06 represents six blanks: the blank specified by >20 and five more. A count of zero characters is valid and allows entry of the DEL character that corresponds to >7F. For example, >20, >7F, >00 represents a blank followed by a DEL character. The character count in bytes 10 and 11 of the call block includes three for each of the examples, not the number of characters displayed as a result of the specified repetition.

When blank adjustment is specified, trailing blanks in the buffer are not written. The write character count in bytes 10 and 11 is not altered.

A Write with Reply operation requires the following in addition to the requirements for a Write ASCII operation:

- The reply flag in the user flags byte set to one
- The extension to the supervisor call block
- The reply block

The Write with Reply option requires the following extension to the basic I/O supervisor call block:

DEC	HEX	
12	C	REPLY BLOCK ADDRESS

2279485

ıe reply block is a three-word block, containing addresses for the Read operation, as follows:

DEC	HEX	
0	0	DATA BUFFER ADDRESS
2	2	READ CHARACTER COUNT
4	4	<ACTUAL READ COUNT>

279486

he three fields are identical to the corresponding fields of the supervisor call block for a Read ıSCII operation.

'he following is an example of the source code for a supervisor call block for a Write ASCII ıperation:

```
WAVDT    DATA 0                    WRITE RECORD TO VDT ASSIGNED TO
         BYTE >B,>4B               LUNO >4B INITIATE MODE.
         BYTE 0,>80
         DATA WRBUFF
         DATA 0
         DATA 80
```

The following is an example of the source code for a supervisor call block for a Write ASCII operation using the Write with Reply option:

```
WRVDT    DATA 0                    WRITE RECORD TO VDT ASSIGNED TO
         BYTE >B,>4B               LUNO >4B INITIATE MODE AND
         BYTE 0,>C0                WRITE WITH REPLY.
         DATA WRBUFF
         DATA 0
         DATA 80
         DATA RBL
```

The reply block is coded as follows:

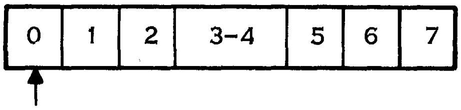
```
RBL      DATA REPLY                REPLY BUFFER ADDRESS
         DATA 80                   MAXIMUM LENGTH OF REPLY
         DATA 0                    REPLY CHARACTER COUNT
```

6.3.2.9 Rewind. Sub-opcode >0E specifies a Rewind operation. The Rewind operation clears the screen of a VDT. Any previously-entered characters that remain in the buffer are ignored.

The following fields of the basic supervisor call block apply to a Rewind operation:

- SVC code — 0
- Return code
- Sub-opcode — >0E
- Logical unit number (LUNO)
- User flags

The following user flag applies to a Rewind operation:



2279487

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

The logical unit number (LUNO) field contains the LUNO assigned to the VDT to be cleared.

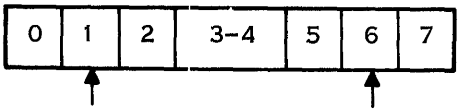
The following is an example of the source code for a supervisor call block to rewind a VDT:

```
RWND    DATA 0                     REWIND VDT ASSIGNED TO LUNO >4A.
        BYTE >E,>4A
        DATA 0
        DATA 0
        DATA 0
        DATA 0
```

6.3.3 VDT Resource-Specific I/O

Most of the resource-specific I/O operations use an extended supervisor call block. The sub-opcodes for the resource-independent operations apply, but the operations are modified by the states of flags in the extended user flags field.

he extended call flag in the user flag field (byte 5) of the supervisor call block must be set to one or resource-specific I/O operations. Otherwise, the system does not use the extensions to the upervisor call block. The flags in the user flag field that apply to resource-specific I/O operations re:



279488

Bit 1 — Reply flag. Set the reply flag to one for a Write with Reply or a Remote Get Event Character. Set the reply flag to zero for all other operations.

Bit 6 — Extended call flag. Set as follows:

1 — Extended call block (required for resource-specific I/O to a VDT).
0 — Basic supervisor call block (used for resource-independent I/O).

The extension to the basic supervisor call block is as follows:

DEC	HEX		
12	C	VALIDATION TABLE/REPLY BLOCK ADDRESS	
14	E	EXTENDED USER FLAGS	
16	10	FILL CHARACTER	<EVENT BYTE>
18	12	CURSOR POSITION ROW	COLUMN
20	14	FIELD BEGINNING ROW	COLUMN

2279489

The extension to the call block contains the following:

Byte	Contents
12–13	Character validation table address (for a Read operation, sub-opcode >09 or >0A, when character validation is specified in the extended user flags). The address of a table of character validation data. Reply block address (for a Write operation, sub-opcode >0B or >0C, when the reply flag is set to one). The address of a block containing the address and count fields for a Write with Reply operation.
14–15	Extended user flags field. Contains 16 flags that apply to all or some of the VDT operations as described in succeeding paragraphs.
16	Fill character. Contains the character to be used to fill character positions when specified.

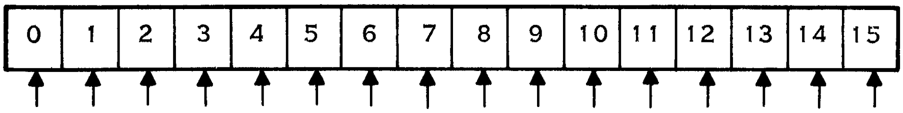
Byte **Contents**

17 <Event byte>. The system stores an event character in this field when the VDT has been opened in the event mode and an event key is pressed. This is the last character entered, which terminates a read call.

18–19 Cursor position: row in byte 18, column in byte 19. Row zero is the top row on the screen; column zero is the leftmost column on the screen. The position to which the cursor is set in a Read operation when the cursor position flag is set to one. The system stores the current cursor position in this field following a Read or Write operation.

20–21 Field beginning definition: row in byte 20, column in byte 21. When the field start position flag is set to one, Read and Write operations begin at this position.

The extended user flags are:



2279490

The following lists the flags and the I/O operations in which they are effective. Detailed descriptions of the uses of the flags follow in subsequent paragraphs.

Bit	Definition	Used in Operations
0	Field start position	All
1	Intensity	All
2	Blink cursor	Read ASCII
3	Graphics	Read/Write
4	Eight-bit ASCII	Read/Write Direct
5	Task edit	Read
6	Beep	Read/Write
7	Right boundary	Read
8	Cursor position within a read field	Read
9	Fill character	Read
10	Do not initialize field	Read
11	Return on termination character	Read
12	No echo	Read
13	Character validation	Read
14	Validation error mode	Read
15	Warning beep	Read

6.3.3.1 Field Start Position. This flag, when set to one, defines the start of the input or output field as specified in bytes 20 and 21 of the extended call block. When this flag is set to zero, the current cursor position defines the start of the field.

A Read ASCII operation with the field start position flag set to one positions the cursor at the beginning of the field (bytes 20 and 21) and begins reading at that point. The cursor position flag and the do not initialize field flag may also be set to one. In that case, previously displayed characters are read up to the cursor position specified in bytes 18 and 19. Additional characters (up to the total specified for the Read operation) are read as entered. At the completion of the Read operation the cursor position is stored in bytes 18 and 19.

For an operation other than Read ASCII, the cursor is moved to the start of the field at the start of the operation.

6.3.3.2 Intensity. This flag, when set to one, specifies high intensity output. When set to zero, the flag specifies low intensity output.

6.3.3.3 Blink Cursor. This flag, when set to one, causes the cursor to blink. When set to zero, the flag causes the cursor to be continuously displayed. The flag applies only to Read ASCII operations.

6.3.3.4 Graphics. This flag, when set to one, causes control characters to be displayed as graphic characters on input and output. Control characters are those represented in ASCII as >00 through >1F. When this flag is set to zero, control characters perform their normal (VDT) functions when entered or written.

6.3.3.5 Eight-Bit ASCII. This flag, when set to one, causes all eight bits of the characters in the buffer to be sent to the terminal during a Write Direct operation. The most significant bit of each character controls the intensity of the display of the character. When bit 0 of the character is a one, the character is displayed with low intensity. When bit 0 is a zero, the character is displayed with high intensity. When the flag is set to zero, only the seven least significant bits of each character are sent to the terminal, and the intensity flag controls the intensity of the display.

During a Read Direct operation with the flag set to one, the intensity of the characters on the screen determines the value of the most significant bit stored in the buffer. The most significant bit of a low-intensity character is set to one; the bit is set to zero for a high-intensity character.

A VDT that supports JISCII characters displays characters with one intensity only.

6.3.3.6 Carriage Control. This flag, when set to one, causes any of the programmable characters listed in Table 6-1 that are entered during an input operation to terminate the operation and to be returned in byte 17 (event byte field) of the extended call block. When the flag is set to zero, the DSR ignores these characters.

6.3.3.7 Beep. This flag, when set to one, causes the terminal to sound an audible tone to request the first character during input operations. During output operations, the tone sounds following display of the last character when the flag is set to one. When the flag is set to zero, the terminal does not sound the audible tone unless the warning beep flag is set.

6.3.3.8 Right Boundary. This flag, when set to one, limits use of the INS CHAR and DEL CHAR keys on the terminal. The flag applies to fields that extend past the right boundary of the current row, and continue on the next row. When the INS CHAR key has been pressed, characters can only be inserted on the current row. The DEL CHAR key is effective only on the current row. When the flag is set to zero, the INS CHAR and DEL CHAR keys are effective across the entire field.

6.3.3.9 Cursor Position. This flag, when set to one, causes a Read ASCII operation to position the cursor to the row and column specified in bytes 18 and 19, respectively, of the extended call block. The Read operation reads previously displayed characters starting at the field start position up to and including the character to the left of the cursor. Additional characters (up to the number specified in bytes 8 and 9 of the call block) are read as entered. The do not initialize field flag determines the cursor position when both flags apply. The system returns the position of the cursor in bytes 18 and 19 at the completion of every Read and Write operation. When the flag is set to zero, the position of the cursor is altered only when the field start position flag is set to one.

6.3.3.10 Fill Character. This flag, when set to one, causes the fill character in byte 16 of the extended call block to be used. The DELETE CHARACTER, ERASE FIELD, ERASE INPUT, and SKIP keys write fill characters in the current field. An initialize field operation also uses the fill character. When the flag is set to zero, a blank is used as the fill character.

6.3.3.11 Do Not Initialize Field. When this flag is set to one, the system does not initialize the current field. When this flag is set to zero, the system initializes the field. Initializing the field consists of writing fill characters in the field and positioning the cursor at the beginning of the field. When the do not initialize field flag is set to zero, the cursor position flag is ignored.

6.3.3.12 Return on Termination Character. This flag, when set to one, causes a Read operation to terminate only when a field termination character is entered. Refer to the information for the appropriate terminal for a list of field termination characters.

When the field has been filled, additional characters are accepted but are neither displayed nor returned to the task. An invalid entry can be corrected by positioning the cursor at a character in error and entering the correct character, or a field termination character may be entered to terminate the operation.

When the flag is set to zero, a Read operation terminates when either the field is full or a field termination character is entered.

6.3.3.13 No Echo. This flag, when set to one, inhibits the display of characters entered at the keyboard. When a key is pressed, the character at the cursor position on the screen is replaced by a blank and the cursor is moved to the next character position. When the flag is set to zero, each character is displayed as it is entered.

6.3.3.14 Character Validation. This flag, when set to one, enables character validation of the field being read by a Read ASCII operation. Character validation is discussed in greater detail in a subsequent paragraph. When the character validation flag is set to zero, no character validation is performed.

6.3.3.15 Validation Error Mode. The validation error mode flag, when set to one, enables correction of errors detected during validation of field contents by the task. The Validation Error Mode operation is effectively a Reread operation; the flags that apply to a read apply in the same way to this operation. The cursor is positioned at the beginning of the previously read field, unless the character position flag is used to position the cursor at some other character. When the user enters the Left arrow, ERASE FIELD, or ERASE INPUT character, the system sets the validation error mode flag to zero. When the calling task sets the validation error mode flag to zero, the operation is performed in the normal mode.

6.3.3.16 Warning Beep. This flag, when set to one, provides an audible warning when certain functions that cannot be performed are requested. Specifically, the warning beep sounds when the following keys are pressed and the indicated conditions apply:

- Left arrow, when cursor is at column one
- DEL CHAR, when there are no characters to delete
- INS CHAR, when there is no more room for characters

6.3.3.17 Examples. To build a display using the graphics available for the VDT, the programmer places graphics codes in a buffer, executes an I/O SVC with the extended call block, and sets the graphics flag to one.

The graphics codes are those in the control code range, >00 through >1F. The graphics symbols that correspond to these codes are shown in Appendix B. The keys that correspond to these codes are listed in Table 6-1.

Table 6-1. Graphics Code Key Equivalents

Graphics Code	911 Key	940 and Business System Key
00	(CONTROL) 3	@
01	(CONTROL) A	!
02	(CONTROL) B	“
03	(CONTROL) C	#
04	(CONTROL) D	$
05	(CONTROL) E	%
06	(CONTROL) F	&
07	(CONTROL) G	'
08	(CONTROL) H	(
09	(CONTROL) I	)
0A	(CONTROL) J	*
0B	(CONTROL) K	+
0C	(CONTROL) L	,
0D	(CONTROL) M	-
0E	(CONTROL) N	.
0F	(CONTROL) O	/
10	(CONTROL) P	0
11	(CONTROL) Q	1
12	(CONTROL) R	2
13	(CONTROL) S	3
14	(CONTROL) T	4
15	(CONTROL) U	5
16	(CONTROL) V	6
17	(CONTROL) W	7
18	(CONTROL) X	8
19	(CONTROL) Y	9
1A	(CONTROL) Z	:
1B	ESC	;
1C	(CONTROL) ,	<
1D	(CONTROL) +	=
1E	(CONTROL) .	>
1F	(CONTROL) /	?

To enter graphics characters, the graphics bit in the IRB must be set.

To enter graphics characters using a 911, simply use the CONTROL key sequence shown in Table 6-1.

To enter graphics characters using a 940 you must first enter the graphics mode by pressing (SHIFT) P2. Press the appropriate key as shown in Table 6-1 to generate the required symbol. Exit the graphics mode by pressing (ALT) 9.

To enter graphics characters using a Business System terminal you must first enter the graphics mode by pressing (ALT) 5 (on the numeric keypad). Press the appropriate key as shown in Table 6-1 to generate the required symbol. Exit the graphics mode by pressing (ALT) 9.

NOTE

> Applications can be written for the 911 that use the key sequences (CONTROL) S and ESC. These sequences *cannot* be used on the 940 and Business System terminals.

The following is an example of an extended supervisor call block for a Write ASCII operation in the graphics mode:

```
WGM     DATA 0               WRITE GRAPHICS DATA TO VDT
        BYTE >B,>4C          AT LUNO >4C. FIELD BEGINS
        DATA >2              AT ROW 0, COL 0. FIELD SIZE
        DATA GBUFF           IS 80 CHARACTERS.
        DATA 0
        DATA 80
        DATA 0
        DATA >9000
        DATA 0
        DATA 0
        DATA 0
```

A Read ASCII operation reads a portion of the field from the screen and a portion from the keyboard by specifying a cursor position within the field. Characters displayed on the screen to the left of the cursor position are read. The remainder of the field is read as the characters are entered.

An example of a Read ASCII operation has the following characteristics:

- The field begins at row 3, column 5.
- The field contains 10 characters.
- The field contains 3 characters to be read.
- Seven or fewer additional characters are to be entered and read.
- A field termination character terminates the operation.
- Additional characters entered are displayed at high intensity.

The following is an example of the code for a supervisor call block for the operation described:

```
RAP      DATA 0          READ FIELD OF VDT AT LUNO >2D.
         BYTE 9,>2D      FIELD SIZE IS 10 CHARACTERS,
         DATA >2         AT ROW 3 COL 5. CHARACTER ENTRY
         DATA RBUFF      BEGINS AT ROW 3 COL 8. SET
         DATA 10         INTENSITY HIGH.
         DATA 0
         DATA 0
         DATA >C0B0
         DATA 0
         BYTE 3
         BYTE 8
         BYTE 3
         BYTE 5
```

When the VDT has been opened in the event key mode and task edit keys are also enabled (carriage control flag set to one), either an event key or a task edit key may terminate a Read operation. The task edit character is always returned in the event byte of the extended call block, and the event character is also returned in that byte in resource-specific I/O. The state of the event key flag in the system flag field indicates which type of character is in the event byte when both are enabled. The task accesses and decodes the character and performs the function corresponding to the key.

The following is an example of the code for a Read ASCII operation with event key termination enabled when the VDT was opened and task edit key termination enabled for the Read operation:

```
RETE     DATA 0          READ FIELD OF VDT AT LUNO >3F.
         BYTE 9,>3F      FIELD SIZE IS 15 CHARACTERS,
SYSFLG   BYTE 0          AT CURRENT CURSOR POSITION.
         BYTE >02        EVENT KEYS AND TASK EDIT KEYS
         DATA RBUFF      ENABLED.
         DATA 15
         DATA 0
         DATA 0
         DATA >0400
         DATA 0
         DATA 0
         DATA 0
```

6.3.3.18 Character Validation Operation. A Read ASCII operation may specify character validation compatible with field validation performed by TIFORM software. After reading a field, the operation only accepts those characters that are within the range or ranges applicable to the operation. Characters that are within the range or ranges are stored in the read buffer specified for the operation. The character validation flag in the extended user flags field is set to one for a Read with Validation operation.

Each Read operation with character validation must specify a validation table. Specifying a validation table requires:

- Setting the character validation flag to one
- Supplying a validation table
- Placing the address of the table in the character validation table address field of the extended call block

The validation table contains one or more ranges of characters that define the valid characters for the field. The table may define the valid characters by specifying ranges of characters that are not valid or by specifying ranges of characters that are valid. Each range in the table requires two bytes, and the table contains two bytes of overhead. Thus the length of the table in bytes is two times the number of ranges, plus two. The format of the table is as follows:

DEC	HEX		
0	0	LENGTH	FLAGS
2	2	RANGE 1 LOW CHAR.	RANGE 1 HIGH CHAR.
2N	2N	RANGE n LOW CHAR.	RANGE n LOW CHAR.

2279491

The validation table contains the following:

Byte	Contents
0	Length — Length of the validation table in bytes (2n + 2).
1	Flags: Bit 0 — Validation flag. Set as follows: 1 — Invalid ranges. Characters greater than or equal to the low character and less than or equal to the high character are invalid. 0 — Valid ranges. Characters greater than or equal to the low character and less than or equal to the high character are valid. Bits 1-7 — Reserved.
2	Low character for range 1.
3	High character for range 1.
	Character pairs for additional ranges.
2n	Low character for range n.
2n + 1	High character for range n.

Character validation is performed after each character is entered in the field, and the cursor is in the position to read the next character. When an invalid character is entered, the beep (audible tone) sounds.

The user must press one of the following correction keys:

- Left CHAR
- ERASE FIELD
- ERASE INPUT
- Left FIELD

Next, the user enters the data correctly.

An example Read with Validation operation performs the following:

- Reads a 10-character field at row 1, column 2.
- Validates the field as an alphanumeric field with no lowercase letters.

The following is an example of the code for the supervisor call block and validation table for the example operation:

```
RASCI    DATA 0                     READ FIELD OF VDT AT LUNO >2B,
         BYTE 9,>2B                 VALIDATING PER TABLE. FIELD
         BYTE 0                     SIZE IS 10 CHARACTERS, AT ROW
         BYTE >02                   1, COL 2. READ BUFFER IS
         DATA BUFF                  BUFF.
         DATA 10
         DATA 0
         DATA TABLE
FLAG2    DATA >8084
         BYTE 1
         BYTE 2
         BYTE 1
         BYTE 2
           .
           .
           .
         EVEN
TABLE    BYTE 6                     LENGTH OF TABLE
         BYTE 0                     VALID RANGES
         DATA >3039                 RANGE 1 — NUMERALS
         DATA >415A                 RANGE 2 — UPPERCASE LETTERS
```

6.3.3.19 Field Validation. Any validation of a field must be performed by a task following the reading of the field. This could verify that the field contains the proper number of letters, followed by numbers, for example. The Read ASCII operation in the validation error mode is used by the task to obtain corrected data when an error has occurred. Character validation and cursor positioning may be requested for the operation also. The validation error mode flag is set to one to enable the mode.

In the validation error mode, the cursor is positioned to read the first character of the field, or at a specified cursor position. The correction keys defined for the character validation operation apply. Entry of a character prior to pressing a correction key causes the beep to sound, the cursor to remain on the same character position, and the displayed character to be unaltered.

The call block for the previous Read ASCII operation can be used by setting the validation error mode flag to one. The following instructions set the flag in the call block of the character validation coding example:

```
MASK2    BYTE  >2
         SOCB  @MASK2,@FLAG2 + 1
```

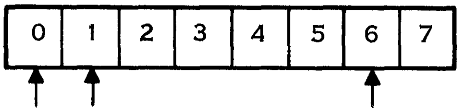
6.3.3.20 Getting Event Characters. The Remote Get Event Character operation (sub-opcode >05) returns an event character in the event byte (byte 17) of the extended supervisor call block. The LUNO assigned to the VDT does not have to be open. The operation is an alternative to performing a Read operation to obtain an event character.

The operation is a special type of Read Device Status operation; the normal read status information is returned.

The following fields of the extended supervisor call block apply to a Remote Get Event Character operation:

- SVC code — 0
- Return code
- Sub-opcode — >05
- Logical unit number (LUNO)
- <System flags>
- User flags
- Data buffer address
- Read character count
- <Actual read count>
- Event byte

The following user flags apply to a Remote Get Event Character operation:



2279492

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

Bit 1 — Reply flag. Set to one.

Bit 6 — Extended call flag. Set to one.

The logical unit number (LUNO) field contains the LUNO assigned to the device at which the event character is entered.

The data buffer address is the address of the buffer into which DNOS places the status information.

The read character count is the length of the buffer.

DNOS returns the event character in the event byte. The event character flag (bit 3 of the system flags) is set to one if an event character is returned.

)NOS maintains an input character queue that stores characters input while the system is)rocessing a previously-entered character or command. Bytes 2 and 3 contain the number of characters in the queue. The size of this queue is specified when the system is generated.

Γhe following is an example of the source code for a supervisor call block for a Remote Get Event Ͻharacter operation and code for the read buffer:

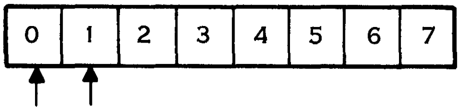
```
RGEVCH   DATA 0              GET EVENT CHARACTER FROM VDT ASSIGNED
         BYTE 5,>32          TO LUNO >32.
         DATA >42
         DATA MRADR
         DATA 18
         DATA 0
         DATA 0
         DATA 0
         BYTE 0
EVCHAR   BYTE 0
         DATA 0,0
MRADR    BSS 1               DEVICE STATUS BUFFER
MCADR    BSS 1
CHINQ    BSS 16
```

6.3.3.21 Read Direct. Sub-opcode >0A specifies a Read Direct operation. The Read Direct operation reads a record from the screen and stores the characters in the specified buffer, two characters per word. The operation begins reading at the character at the cursor position.

The following fields of the basic supervisor call block apply to a Read Direct operation:

- SVC code — 0
- Return code
- Sub-opcode — >0A
- Logical unit number (LUNO)
- System flags
- User flags
- Data buffer address
- Read character count
- <Actual read count>

The following system flags apply to a Read Direct operation:

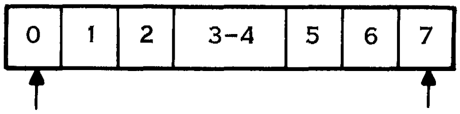


2279493

Bit 0 — Busy flag. Set by system as follows:
1 — Busy.
0 — Operation completed.

Bit 1 — Error flag. Set by system as follows:
1 — Error.
0 — No error.

The following user flags apply to a Read Direct operation:



2279494

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

Bit 7 — Blank adjustment flag. Set as follows:
1 — Read with blank adjustment.
0 — Read without blank adjustment.

The logical unit number (LUNO) field contains the LUNO assigned to the VDT from which a record is to be read.

The data buffer address is the address of the buffer into which DNOS places the record.

The read character count is the length of the buffer.

DNOS returns the number of characters stored in the buffer in the actual read count field.

The Read Direct operation recognizes the ASCII or JISCII codes as specified by the country code. The operation stores the characters in a word, packed one per byte. The most significant bit is set to zero for the seven-bit ASCII codes; all eight bits of the JISCII code are stored. The operation continues until the buffer is full. DNOS places the number of characters stored in the buffer in the actual read count field.

The following is an example of the source code for a supervisor call block for a Read Direct opera-ion and code for the read buffer:

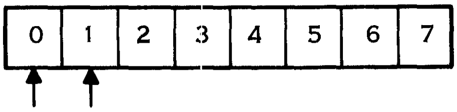
```
RDDVDT    DATA 0                        READ RECORD FROM VDT ASSIGNED TO
          BYTE >A,>3F                   LUNO >3F.
          BYTE 0,0
          DATA RDBUFF
          DATA 80
          DATA 0
RDBUFF    BSS 80                        READ BUFFER
```

6.3.3.22 Write Direct. Sub-opcode >0C specifies a Write Direct operation. The Write Direct operation transfers a record from the specified buffer to the screen of the VDT.

The following fields of the basic supervisor call block apply to a Write Direct operation:

- SVC code — 0
- Return code
- Sub-opcode — >0C
- Logical unit number (LUNO)
- System flags
- User flags
- Data buffer address
- Write character count

The following system flags apply to a Write Direct operation:

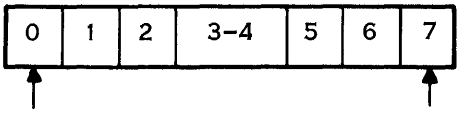


2279495

Bit 0 — Busy flag. Set by system as follows:
1 — Busy.
0 — Operation completed.

Bit 1 — Error flag. Set by system as follows:
1 — Error.
0 — No error.

The following user flags apply to a Write Direct operation:



2279496

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

Bit 7 — Blank adjustment flag. Set as follows:
1 — Write with blank adjustment.
0 — Write without blank adjustment.

The logical unit number (LUNO) field contains the LUNO assigned to the VDT to which a record is to be written.

The data buffer address is the address of the buffer that contains the record to be displayed.

The write character count is the number of characters to be displayed on the screen of the VDT.

The Write Direct operation displays a record on the screen of a VDT. The record consists of the seven least significant bits of each byte in the buffer.

When blank adjustment is specified, trailing blanks in the buffer are not written. The write character count in bytes 10 and 11 is not altered.

The following is an example of the source code for a supervisor call block for a Write Direct operation:

```
WAVDT    DATA 0              WRITE RECORD TO VDT ASSIGNED TO
         BYTE >C,>4B         LUNO >4B INITIATE MODE.
         BYTE 0,>80
         DATA WRBUFF
         DATA 0
         DATA 80
```

6.3.4 VDT Terminal Specific Information

The following paragraphs describe the unique characteristics of the 911, 931, and 940 VDTs as they apply to VDT I/O.

Figure 6-1, Figure 6-2, Figure 6-3, and Figure 6-4 show the 911, 931, 940 VDT, and Business System Terminal keyboards. Table 6-2 lists the programmable keys on the VDT keyboards, showing the codes returned by the device service routines (DSRs) for each key. The terminal code column lists the ASCII codes and control character designations that correspond to these keys.

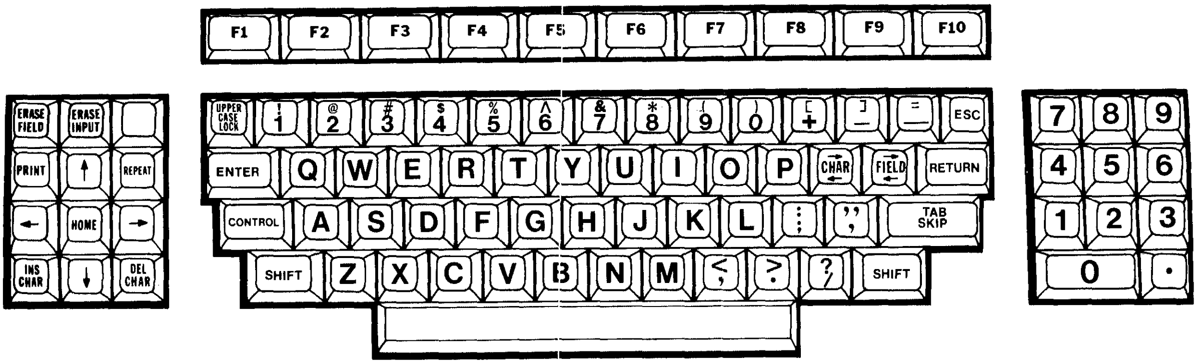

F1 F2 F3 F4 F5 F6 F7 F8 F9 F10
ERASE FIELD ERASE INPUT
PRINT REPEAT
HOME
INS CHAR DEL CHAR
UPPER CASE LOCK ! 1 @ 2 # 3 $ 4 % 5 ∧ 6 & 7 * 8 (9) 0 [+] — = — ESC
ENTER Q W E R T Y U I O P CHAR FIELD RETURN
CONTROL A S D F G H J K L : ; " ' TAB SKIP
SHIFT Z X C V B N M < , > . ? / SHIFT
7 8 9 4 5 6 1 2 3 0 .
2283185

Figure 6-1. Model 911 VDT Keyboard

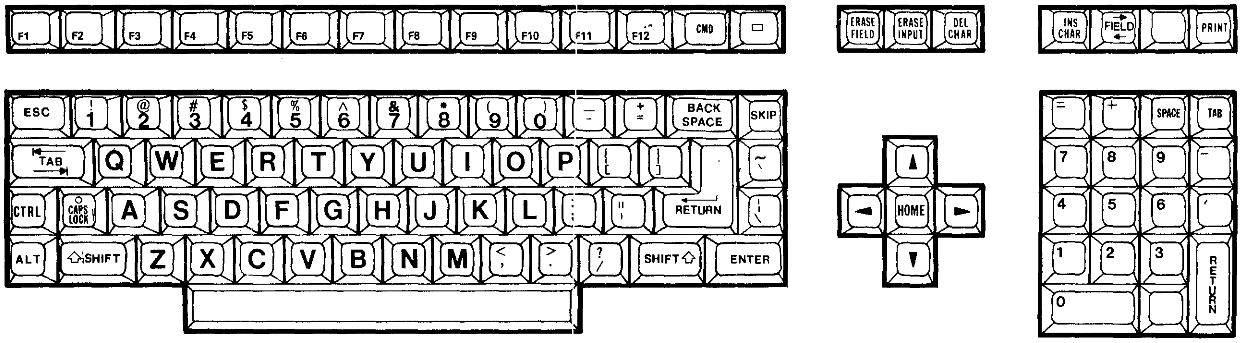
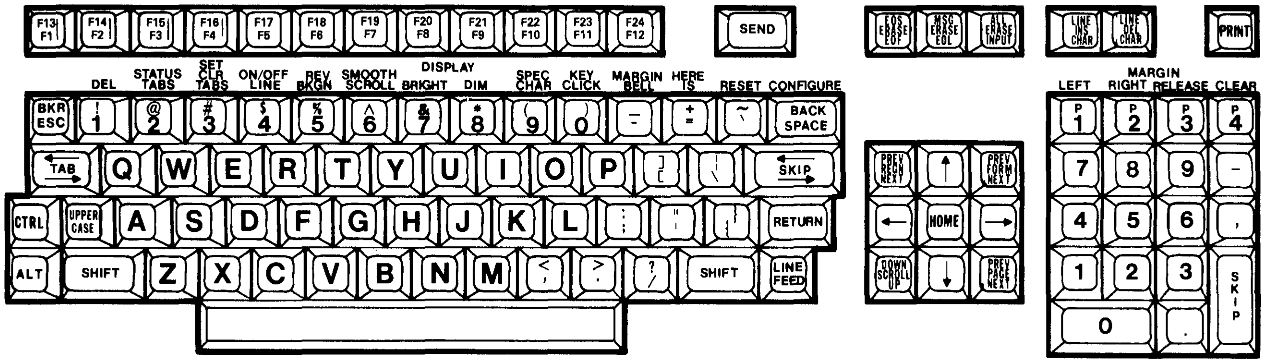
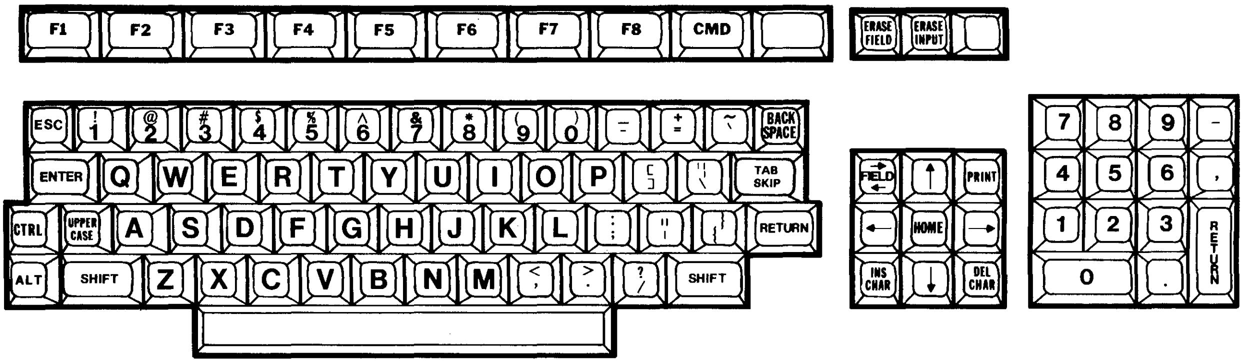

F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12 CMD
ERASE FIELD ERASE INPUT DEL CHAR
INS CHAR FIELD PRINT
ESC ! 1 @ 2 # 3 $ 4 % 5 ∧ 6 & 7 * 8 (9) 0 _ - + = BACK SPACE SKIP
TAB Q W E R T Y U I O P { [}] ~ `
CTRL CAPS LOCK A S D F G H J K L : ; " ' RETURN | \
ALT SHIFT Z X C V B N M < , > . ? / SHIFT ENTER
HOME
= + SPACE TAB
7 8 9 -
4 5 6 ,
1 2 3 RETURN
0
2284204

Figure 6-2. Model 931 VDT Keyboard



2283186

Figure 6-3. Model 940 VDT Keyboard



2283106

Figure 6-4. Business System Terminal Keyboard

Table 6-2. Terminal Key Designations and Codes

DSR Code[1]	Type[2]	Business System Key	911 Key	931 Key	940 Key
7F	S	ERASE FIELD	ERASE FIELD	ERASE FIELD	ERASE EOF
80	E	SHIFT F1	CONTROL 1	F9	F9
81	E	F1	F1	F1	F1
82	E	F2	F2	F2	F2
83	E	F3	F3	F3	F3
84	E	F4	F4	F4	F4
85	E	F5	F5	F5	F5
86	E	F6	F6	F6	F6
87	T	Right FIELD	Right FIELD	Right FIELD	LINE FEED
88	S	Left Arrow	Left Arrow	Left Arrow	Left Arrow
89	ST	TAB	TAB	TAB	TAB Right
8A	T	Down Arrow	Down Arrow	Down Arrow	Down Arrow
8B	ST	SKIP	SKIP	SKIP	SKIP Right
8C	ST	HOME	HOME	HOME	HOME
8D	STR	RETURN	RETURN	RETURN	RETURN
8E	ST	ERASE INPUT	ERASE INPUT	ERASE INPUT	ERASE INPUT
8F	T	Blank Gray	Blank Gray	Blank Gray	INS LINE
90	S	DEL CHAR	DEL CHAR	DEL CHAR	DEL CHAR
91	S	INS CHAR	INS CHAR	INS CHAR	INS CHAR
92	S	Right Arrow	Right Arrow	Right Arrow	Right Arrow
93	TRF	ENTER	ENTER	ENTER	SEND
94	T	Left FIELD	Left FIELD	Left Field	SKIP Left
95	T	Up Arrow	Up Arrow	Up Arrow	Up Arrow
96	E	F7	F7	F7	F7
97	E	F8	F8	F8	F8
98	E	CMD	CMD	CMD	NEXT FORM
99	E	PRINT	PRINT	PRINT	PRINT
9A	E	SHIFT F2	CONTROL 2	F10	F10
9B	H	Blank Orange	Blank Orange	Blank Orange	PREV FORM
9C	E	SHIFT F3	CONTROL 4	F11	F11
9D	E	SHIFT F4	CONTROL 5	F12	F12
9E	E	SHIFT F5	CONTROL 6	SHIFT F1	F13
9F	E	SHIFT F6	CONTROL 7	SHIFT F2	F14

Notes:

[1] Codes in this column are shown as hexadecimal numbers.

[2] Letters in this column denote the following:
S — System edit key; E — Event key; T — Task edit key; H — Hold key; F — End-of-file; R — End-of-record.

The VDTs that support JISCII characters display characters in one intensity only.

The field termination characters for the VDTs are the RETURN and ENTER Keys.

6.3.5 VDT Read Device Status Operation

The Read Device Status operation returns four bytes of status information for VDTs. The first and second bytes contain the maximum row and column addresses for the VDT. The third and fourth bytes contain the number of keyboard input characters that are currently internally buffered in the character queue. This information is returned in bytes zero through three for the 911, 940, and Business Systems terminals.

The 911 VDT returns a total of 18 bytes of information. The 931, 940, and the Business Systems terminals return 16 additional bytes of information, provided the buffer has sufficient space.

Byte	Meaning
0	Maximum row address (24)
1	Maximum column address (80)
2,3	Number of keyboard characters in input character queue
4	DSR type: >11 = 911 >31 = 931/940 (DNOS version 1.2) >40 = 940 or Business Systems (earlier DNOS versions)
5	Reserved (>00)
6,7	CRU address
8,9	Reserved (>FFFF)
10	Interface type: >01 = Communications interface for 911 >06 = S300 EVDT port >07 = 990/10A 9902 port >08 = CI402 9902 port >09 = CI421 9902 port >0A = CI422 9902 port >23 = CI403 port >24 = CI404 port >30 = CI421 9903 port
11	Port ID number for 931/940 Always >11 for 911

Byte	Meaning
12–13	Edit flag word 1
14–15	Edit flag word 2
16–21	Reserved
22	Line flags
	0 = half duplex
	1 = switched line
	2–7 = Reserved (0)
23	Reserved
24	Speed code
	>00 = 50 baud
	>02 = 110 baud
	>03 = 134.5 baud
	>04 = 150 baud
	>05 = 200 baud
	>06 = 300 baud
	>07 = 600 baud
	>08 = 1200 baud
	>09 = 1800 baud
	>0A = 2400 baud
	>0B = 3600 baud
	>0C = 4800 baud
	>0D = 7200 baud
	>0E = 9600 baud
	>0F = 14400 baud
	>10 = 19200 baud
	>11 = 28800 baud
	>12 = 34800 baud
25–35	Reserved
36	Terminal type:
	>A1 = 931
	>B0 = 940

6.4 733 ASR DATA TERMINAL I/O

DNOS supports both resource-independent and resource-specific I/O for the 733 ASR data terminal. The keyboard/printer portion of the 733 ASR is included here. The cassette units of the 733 ASR are described in a subsequent paragraph. Resource-independent I/O for these terminals includes operations that are analogous to sequential file operations.

Resource-specific I/O for the 733 ASR data terminal includes operations that apply only to the 733 ASR. These operations give a program control over functions of special keys. Except for the Read Device Status operation, the device must be opened using sub-opcode >00 or >03 prior to any I/O operation.

The following I/O call block for 733 ASR operations is the basic block used for all I/O operations. If an extension to this block is necessary for a particular operation, it is indicated in the operation description.

SVC >00 -- I/O Operations

Align on word boundary
Can be initiated as an event

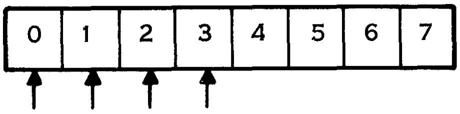
Dec	Hex		
0	0	>00	<Return Code>
2	2	Sub-Opcode	LUNO
4	4	<System Flags>	User Flags
6	6	Data Buffer Address	
8	8	Read Character Count	
10	A	Write Character Count/ < Actual Read Count>	

2279470

The subset of sub-opcodes for the 733 ASR applies to both resource-independent and resource-specific I/O, as follows:

00	Open
01	Close
02	Close, Write EOF
03	Open and Rewind
04	Close and Unload
05	Read Device Status
09	Read ASCII
0B	Write ASCII
0D	Write EOF

The system flags (byte 4) in the supervisor call block apply to all 733 ASR data terminal I/O. These flags are:



2279497

Bit 0 — Busy flag. Set by system as follows:
1 — Busy.
0 — Operation completed.

Bit 1 — Error flag. Set by system as follows:
1 — Error.
0 — No error.

Bit 2 — End-of-file. Set by system as follows:
1 — (CTRL) S key terminated the operation.
0 — Operation terminated without the (CTRL) S key being pressed.

Bit 3 — Event key flag. Set by system as follows:
1 — An event key terminated the operation.
0 — Operation terminated without an event key being pressed.

The user flags (byte 5) in the supervisor call block apply to all 733 ASR data terminal I/O. However, significance of these flags differs for various operations. The flags that apply to each operation are described in the detailed description of each operation.

The character set applicable to a hard-copy terminal is the ASCII character set or the JISCII character set appropriate to the country code for which the system was generated.

6.4.1 Key Categories

The system interprets each key on the keyboard in one or more of the following categories:

- Data
- Hold
- Event
- System edit
- Task edit

Table 6-3 lists the keys on 733 ASR data terminal keyboards and the type of each key. The DSR code column shows the codes returned by the device service routine (DSR) for each key. The terminal code column lists the ASCII codes and control character designations that correspond to these keys. The Device Character Set appendix also contains the information in Table 6-3.

Table 6-3. 733 ASR Key Designations and Codes for ASCII Mode

DSR Code[1]	Type[2]	Terminal Code[1]	Key
7F	S	7F DEL	RUB OUT
80	E	00 NULL	(CTRL) 3
81	E	01 SOH	(CTRL) A
82	E	02 STX	(CTRL) B
83	E	03 ETX	(CTRL) C
84	E	04 EOT	(CTRL) D
85	E	05 ENQ	(CTRL) E
86	E	06 ACK	(CTRL) F
87	T	07 BELL	(CTRL) G
88	S	08 BS	(CTRL) H
89	ST	09 HT	(CTRL) I
8A	T	0A LF	(CTRL) J or LINE FEED
8B	ST	0B VT	(CTRL) K
8C	ST	0C FF	(CTRL) L
8D	STR	0D CR	(CTRL) M or RETURN
8E	ST	0E SO	(CTRL) N
8F	T	0F SI	(CTRL) O
90	S	10 DLE	(CTRL) P
91	S	11 DC1	(CTRL) Q
92	S	12 DC2	(CTRL) R
93	TRF	13 DC3	(CTRL) S
94	T	14 DC4	(CTRL) T
95	T	15 NAK	(CTRL) U
96	E	16 SYN	(CTRL) V
97	E	17 ETB	(CTRL) W
98	E	18 CAN	(CTRL) X
99	E	19 EM	(CTRL) Y
9A	E	1A SUB	(CTRL) Z
9B	H	1B ESC	ESC
9C	E	1C FS	(CTRL) ,
9D	E	1D GS	(CTRL) _
9E[3]	E	1E RS	(CTRL) .
9F[3]	E	1F US	(CTRL) /

Notes:

[1] Codes in this column are shown as hexadecimal numbers.

[2] Letters in this column denote the following:
S — System edit key; E — Event key; T — Task edit key; H — Hold key; F — End-of-file; R — End-of-record.

[3] DSR codes >9E and >9F are not available on 733 ASR data terminals that support JISCII. Keys (CTRL) . and (CTRL) / generate codes >8E and >8F, respectively.

6.4.1.1 Data Keys. The data keys return the codes of printable characters to the buffer specified in the call block. The category includes the keys that return ASCII codes >20 through >7E.

6.4.1.2 Hold Key. The hold key suspends output to the terminal. Operation may be resumed by pressing any other key except RETURN, exclamation point (!), or (CTRL) X. The hold key is the ESC key.

When the RETURN key is pressed following the hold key, the current operation is aborted, and an error code is returned to the task that requested the I/O.

When the exclamation point key is pressed following the hold key, the system activates SCI and the output continues. Contention between the interrupted output and SCI for the use of the terminal may cause unpredictable results.

When (CTRL) X is pressed following the hold key, a hard break results. The hard break terminates the current task and activates SCI. The end action specified by the task, if any, is performed prior to terminating the task and activating SCI. The hard break should be used to abort tasks when appropriate; it should be used with care because the hard break aborts pending I/O requests.

The system selects the task to be terminated by the hard break from among the tasks of the current job. The following rules apply in the listed order of priority:

1. When only the SCI task is active, SCI is terminated.
2. Any other foreground task is terminated.
3. Any other background task is terminated.
4. When other tasks are active along with SCI, SCI is terminated last.

There may be times when the hard break terminates a task other than the one intended. In that case, press the hold key and (CTRL) X again to terminate the intended task.

The effect of a subsequent hard break depends upon the timing. When the task has not yet taken end action in response to the first hard break, the subsequent hard break terminates the next task in the order of priority. When the task is executing the end action routine but has not completed end action, the task is aborted as if end action had not been provided. When end action has completed, the subsequent hard break causes the task to take end action again.

The end action routine should execute a Get End Action Status SVC to obtain the error code and identify the cause of the termination. When the task error code returned by the SVC is >10, a hard break has occurred. The task may process the hard break and resume execution after executing a Reset End Action Status SVC. The task must place the WP, PC, and ST values returned by the Get End Action Status SVC in R13, R14, and R15 and execute an RTWP instruction to resume execution.

6.4.1.3 Event Keys. Activating the event key mode enables use of event keys as task programmable function keys. The event key mode is activated by performing an Open operation with the event key mode bit (bit 7 of byte 5 of the call block) set to one. Event characters may be accessed by a Remote Get Event Character operation without opening the LUNO assigned to the terminal.

When an event key is pressed, the corresponding character code is stored in the event character buffer. When an input operation using an extended call block is being performed, the operation terminates with the event key bit (bit 3 of byte 4 of the call block) set to one.

When no input operation is being performed and an event key is pressed, the next input operation is immediately terminated with the event key bit of the call block set to one.

The event keys are:

(CTRL) 3
(CTRL) A
(CTRL) B
(CTRL) C
(CTRL) D
(CTRL) E
(CTRL) F
(CTRL) V
(CTRL) W
(CTRL) X
(CTRL) Y
(CTRL) Z
(CTRL) ,
(CTRL) _
(CTRL) .
(CTRL) /

The task decodes the event character and performs the desired function. When the input operation that terminates with the event key bit set to one uses the extended call block (resource-specific I/O), the event character is returned in the event character field. The event character can be obtained without performing a read operation (and without opening the LUNO) by performing a Remote Get Event Character operation. When the input operation uses the basic I/O call block (resource-independent I/O), the event character is not returned.

NOTE

> Any task may access the event character buffer by performing I/O to any LUNO assigned to the terminal. The first access is the only access that returns the correct character. Tasks that perform I/O to a terminal to which SCI is performing I/O must avoid accessing event characters. Either the task or SCI may fail to perform the intended function.

6.4.1.4 System and Task Edit Keys. System edit keys are control keys that are implemented by the system. Task edit keys are control keys that are task functions. Five of the keys are both system and task edit keys, for which the system performs a function. The task may perform an additional function.

Task edit functions do not apply to resource-independent I/O. The carriage control bit in the extended user flags field must be set to one to enable task edit functions. When task edit functions are enabled, pressing a task edit key during an Input operation terminates the operation. The device service routine (DSR) returns the character code of the task edit character in the event byte field of the extended call block.

The following paragraphs describe the edit keys. The paragraph that discusses the system edit key describes the function performed when the key is pressed. For task edit keys, the paragraphs state the code that the DSR returns to the task when the keys are pressed. For keys that are both system and task edit, the paragraphs describe the functions and state the codes. System edit keys are listed with a letter S in the type column of Table 6-3; task edit keys are listed with a letter T in the type column.

Edit keys are related to fields of data on the screen, which are read in an I/O operation. The field begins at the cursor position and consists of the number of characters to be read by the current read operation. When an edit key is pressed between read operations, it is effective at the beginning of the next read operation.

RUB OUT Key. This system edit key performs a carriage return followed by a line feed.

(CTRL) O Key. This task edit key returns the character code >8F in the event byte field.

(CTRL) U Key. This task edit key returns the character code >95 in the event byte field.

(CTRL) S Key. This task edit key returns the character code >93 in the event byte field. The device service routine (DSR) interprets this code as end-of-file (EOF) and sets bit 2 of byte 4 of the call block.

(CTRL) T Key. This task edit key returns the character code >94 in the event byte field.

(CTRL) G Key. This task edit key returns the character code >87 in the event byte field.

(CTRL) J Key. This task edit key returns the character code >8A in the event byte field.

(CTRL) N Key. This system and task edit key fills the field with the fill character and returns the character code >8E in the event byte field.

RETURN Key. This system and task edit key performs a carriage return function and returns the character code >8D in the event byte field.

(CTRL) L Key. This system and task edit key returns the character code >8C in the event byte field.

(CTRL) K Key. This system and task edit key returns the character code >8B in the event byte field.

(CTRL) I Key. This system and task edit key returns the character code >89 in the event byte field.

6.4.2 733 ASR Data Terminal Resource-Independent I/O

The operations appropriate for the 733 ASR data terminals are described in subsequent paragraphs. The following sub-opcodes, which do not apply to 733 ASR data terminals, produce the indicated results:

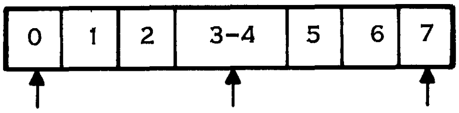
06	Ignored
07	Ignored
08	Error
0E	Ignored
0F	Ignored

6.4.2.1 Open. Sub-opcode >00 specifies an Open operation. The Open operation causes the terminal to perform a line feed and a carriage return and is required for 733 ASR data terminals.

The following fields of the basic supervisor call block apply to an Open operation:

- SVC code — 0
- Return code
- Sub-opcode — >00
- Logical unit number (LUNO)
- User flags
- Data buffer address
- Read character count

The following user flags apply to an Open operation:



2279498

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

Bits 3-4 — Access privilege flag. Set as follows:
00 — Exclusive write.
01 — Exclusive all.
10 — Shared.
11 — Read only.

Bit 7 — Event key mode flag. Set as follows:
1 — Enable event key mode.
0 — Disable event key mode.

The logical unit number (LUNO) field contains the LUNO assigned to the 733 ASR data terminal to be opened.

The Open operation returns the device type code in the data buffer address field (bytes 6 and 7) of the supervisor call block. The device type code for the 733 ASR data terminal is 1.

When the calling task places zero in the read character count field (bytes 8 and 9) of the supervisor call block, the Open operation returns the default logical record length for the device. The default logical record length for a 733 ASR data terminal is >56.

A 733 ASR data terminal must be opened with the event key flag set to one if event keys are to be used as task programmable function keys.

To access an event key character, perform a Remote Get Event Character operation.

The following is an example of the source code for a supervisor call block to open a 733 ASR data terminal:

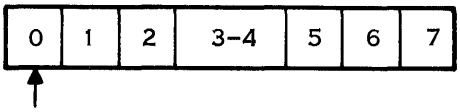
```
OHCT     DATA 0                  OPEN TERMINAL ASSIGNED TO LUNO >20.
         BYTE 0,>20
         DATA 0
TPE      DATA 0
LRL      DATA 0
         DATA 0
```

6.4.2.2 Close. Sub-opcode >01 specifies a Close operation. The Close operation ends I/O to a LUNO from the calling task. The LUNO remains assigned to the device, and may be opened again for additional I/O operations. When a task terminates, DNOS closes all LUNOs that have been opened by the task.

The following fields of the basic supervisor call block apply to a Close operation:

- SVC code — 0
- Return code
- Sub-opcode — >01
- Logical unit number (LUNO)
- User flags

The following user flag applies to a Close operation:



2279499

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

The logical unit number (LUNO) field contains the LUNO to be closed.

The following is an example of the source code for a supervisor call block to close a 733 ASR data terminal:

```
CHCT     DATA 0                  CLOSE TERMINAL ASSIGNED TO LUNO >20.
         BYTE 1,>20
         DATA 0
         DATA 0
         DATA 0
         DATA 0
```

6.4.2.3 Close, Write EOF. The Close, Write EOF operation, sub-opcode >02, performs three line feed operations on the 733 ASR data terminal, followed by a Close operation.

6.4.2.4 Open and Rewind. The Open and Rewind operation, sub-opcode >03, is an Open operation followed by a Rewind operation. For the 733 ASR data terminals, the Rewind operation consists of clearing the input character queue. The Open and Rewind operation causes the terminal to perform a line feed and a carriage return.

6.4.2.5 Close and Unload. The Close and Unload operation, sub-opcode >04, is identical to the Close operation.

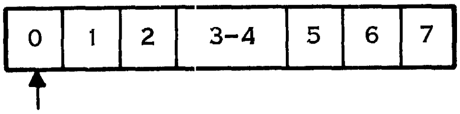
6.4.2.6 Read Device Status. Sub-opcode >05 specifies a Read Device Status operation. The Read Device Status operation returns the number of characters currently stored in the input character queue.

The following fields of the basic supervisor call block apply to a Read Device Status operation:

- SVC code — 0
- Return code
- Sub-opcode — >05
- Logical unit number (LUNO)
- User flags

- Data buffer address
- Read character count
- <Actual Read Count>

The following user flag applies to a Read Device Status operation:



2279500

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

The logical unit number (LUNO) field contains the LUNO assigned to the device for which status information is returned.

The data buffer address is the address of the buffer into which DNOS places the status information.

The read character count is the length of the buffer.

DNOS returns the number of characters stored in the buffer in the actual read count field. The system returns >0004 in this field when the specified LUNO is assigned to a 733 ASR data terminal.

After a Read Device Status operation returns the status of a 733 ASR data terminal the data buffer contains the following:

Byte	**Contents**
0	>FF.
1	>FF.
2–3	Number of characters currently stored in the input character queue.

DNOS maintains an input character queue that stores characters input while the system is processing a previously-entered character or command. Bytes 2 and 3 contain the number of characters in the queue. The maximum size of this queue is specified when the system is generated.

The following is an example of the source code for a supervisor call block for a Read Device Status operation and code for the read buffer:

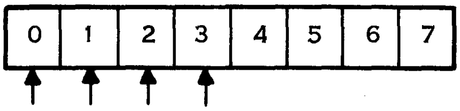
```
RDSHCT   DATA 0                      READ STATUS OF TERMINAL ASSIGNED TO
         BYTE 5,>35                  LUNO >35.
         DATA 0
         DATA DMY
         DATA 10
         DATA 0
DMY      BSS 2                       DEVICE STATUS BUFFER
LIQ      BSS 8
```

6.4.2.7 Read ASCII. Sub-opcode >09 specifies a Read ASCII operation. The Read ASCII operation reads a record from the keyboard and stores the characters in the specified buffer, two characters per word.

The following fields of the basic supervisor call block apply to a Read ASCII operation:

- SVC code — 0
- Return code
- Sub-opcode — >09
- Logical unit number (LUNO)
- <System flags>
- User flags
- Data buffer address
- Read character count
- <Actual read count>

The following system flags apply to a Read ASCII operation:



2279501

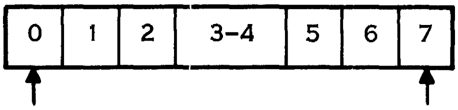
Bit 0 — Busy flag. Set by system as follows:
1 — Busy.
0 — Operation completed.

Bit 1 — Error flag. Set by system as follows:
1 — Error.
0 — No error.

Bit 2 — End-of-file. Set by system as follows:
1 — (CTRL) S key terminated the operation.
0 — Operation terminated without the (CTRL) S key being pressed.

Bit 3 — Event key flag. Set by system as follows:
1 — An event key terminated the operation.
0 — Operation terminated without an event key being pressed.

The following user flags apply to a Read ASCII operation:



2279502

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

Bit 7 — Blank adjustment flag. Set as follows:
1 — Read with blank adjustment.
0 — Read without blank adjustment.

The logical unit number (LUNO) field contains the LUNO assigned to the terminal from which a record is to be read.

The data buffer address is the address of the buffer into which DNOS places the record.

The read character count is the length of the buffer.

DNOS returns the number of characters stored in the buffer in the actual read count field.

The Read ASCII operation recognizes the characters listed in Appendix B for the 733 ASR data terminal. The operation stores the characters, packed one per byte. When the country code in effect is not >0200 (Japan), the most significant bit is set to zero. When the country code is >0200, the eight-bit JISCII code is stored. The operation continues until the RETURN key is pressed, the buffer is full, or (if the terminal is in the event key mode) an event key is pressed.

Characters that are entered between Read operations are stored in a queue and read by the next Read operation. The maximum size of the queue is specified when the system is generated. Additional characters entered after the queue is full are ignored.

Characters can be corrected by pressing the (CTRL) H or BACKSPACE key. The terminal performs a backspace operation and deletes the previously entered character from the data buffer each time the key is pressed. The first time the key is pressed, the printer also performs a line feed operation. After spacing to the character in error, reenter the characters deleted.

When the RETURN key is pressed, the number of characters entered is stored in the actual read count field and the operation terminates.

When the terminal was opened in the event key mode and an event key is pressed, the system sets the event key flag in the system flags byte and terminates the operation. The event character may be accessed by performing a Remote Get Event Character operation or a Get Event Character SVC.

When an event key is pressed between Read operations, the next Read operation performed after the pressing of the event key terminates with the event key flag set and zero in the actual read count field.

When the (CTRL) S key is pressed, the system sets the EOF flag in the system flags byte and terminates the operation.

When blank adjustment is specified for variable length records, blanks are stored in the buffer to fill the record. That is, when the record length is less than the buffer size, the device service routine (DSR) supplies blanks (>20) to fill the buffer. The character count returned in bytes 10 and 11 includes the blanks supplied by the DSR.

The following is an example of the source code for a supervisor call block for a Read ASCII operation and code for the read buffer:

```
RDHCT    DATA 0                   READ RECORD FROM TERMINAL ASSIGNED
         BYTE 9,>2C               TO LUNO >2C IN THE INITIATE I/O MODE.

         BYTE 0,>80
         DATA RBUF
         DATA 80
         DATA 0
RBUF     BSS 80                   READ BUFFER
```

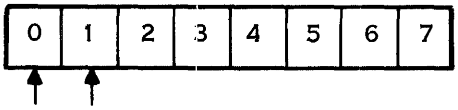
6.4.2.8 Write ASCII. Sub-opcode >0B specifies a Write ASCII operation. The Write ASCII operation transfers a record from the specified buffer to the terminal. DNOS supports a Write with Reply option, which is effectively a Write operation followed by a Read ASCII operation.

The following fields of the basic supervisor call block apply to a Write ASCII operation:

- SVC code — 0
- Return code
- Sub-opcode — >0B
- Logical unit number (LUNO)

- <System flags>
- User flags
- Data buffer address
- Write character count
- Reply block address (write with reply option)

The following system flags apply to a Write ASCII operation:

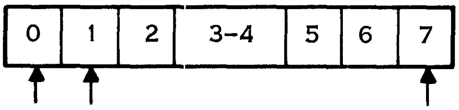


2279503

Bit 0 — Busy flag. Set by system as follows:
1 — Busy.
0 — Operation completed.

Bit 1 — Error flag. Set by system as follows:
1 — Error.
0 — No error.

The following user flags apply to a Write ASCII operation:



2279504

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

Bit 1 — Reply flag. Set as follows:
1 — Write operation followed by a Read operation.
0 — All other operations.

Bit 7 — Blank adjustment flag. Set as follows:
1 — Write with blank adjustment.
0 — Write without blank adjustment.

The logical unit number (LUNO) field contains the LUNO assigned to the terminal to which a record is to be written.

The data buffer address is the address of the buffer that contains the record to be printed.

The write character count is the number of characters to be printed by the terminal.

The Write ASCII operation prints a record on the terminal. The record consists of ASCII characters or JISCII characters, as specified by the country code.

When the HT character (>09) is transferred to the terminal, the result is a space. When the Form Feed character (>0C) is transferred, the printer performs eight line feed operations.

When blank adjustment is specified, trailing blanks in the buffer are not written. The write character count in bytes 10 and 11 is not altered.

A Write with Reply operation requires the following in addition to the requirements for a Write ASCII operation:

- The reply flag in the user flags byte set to one
- The extension to the supervisor call block
- The reply block

The extension to the basic I/O supervisor call block is as follows:

DEC	HEX	
12	0	REPLY BLOCK ADDRESS

2279505

The reply block is a three-word block, containing addresses for the Read operation, as follows:

DEC	HEX	
0	0	DATA BUFFER ADDRESS
2	2	READ CHARACTER COUNT
4	4	<ACTUAL READ COUNT>

2279506

The three fields are identical to the corresponding fields of the supervisor call block for a Read ASCII operation.

The following is an example of the source code for a supervisor call block for a Write EOF operation:

```
WAHCT   DATA 0              WRITE RECORD TO TERMINAL ASSIGNED TO
        BYTE >B,>4C         LUNO >4C INITIATE MODE.
        BYTE 0,>80
        DATA WRBUFF
        DATA 0
        DATA 80
```

The following is an example of the source code for a supervisor call block for a Write ASCII operation using the Write with Reply option:

```
WRHCT     DATA 0                      WRITE RECORD TO TERMINAL ASSIGNED TO
          BYTE >B,>4C                 LUNO >4C INITIATE MODE AND
          BYTE 0,>C0                  WRITE WITH REPLY.
          DATA WRBUFF
          DATA 0
          DATA 80
          DATA RBK
```

The reply block is coded as follows:

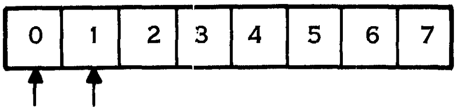
```
RBK       DATA REPLY                  REPLY BUFFER ADDRESS
          DATA 80                     MAXIMUM LENGTH OF REPLY
          DATA 0                      REPLY CHARACTER COUNT
```

6.4.2.9 Write EOF. The Write EOF operation (sub-opcode >0D) performs three line feed operations on a 733 ASR data terminal.

The following fields of the basic supervisor call block apply to a Write EOF operation:

- SVC code — 0
- Return code
- Sub-opcode — >0D
- Logical unit number (LUNO)
- System flags
- User flags

The following system flags apply to a Write EOF operation:

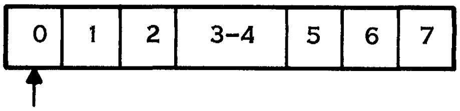


2279507

Bit 0 — Busy flag. Set by system as follows:
1 — Busy.
0 — Operation completed.

Bit 1 — Error flag. Set by system as follows:
1 — Error.
0 — No error.

The following user flags apply to a Write EOF operation:



2279508

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

The logical unit number (LUNO) field contains the LUNO assigned to the terminal to which a record is to be written.

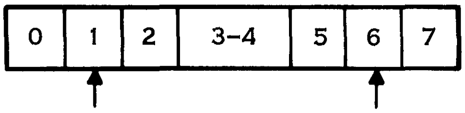
The following is an example of the source code for a supervisor call block for a Write ASCII operation:

```
WEHCT   DATA 0               WRITE EOF TO TERMINAL ASSIGNED TO
        BYTE >D,>4C          LUNO >4C INITIATE MODE.
        BYTE 0,>80
        DATA 0
        DATA 0
        DATA 0
```

6.4.3 733 ASR Data Terminal Resource-Specific I/O

Most of the resource-specific I/O operations use an extended supervisor call block. The sub-opcodes for the resource-independent operations apply, but the operations are modified by the states of flags in the extended user flags field. Sub-opcode >0A (Read Direct operation) does not apply to the 733 ASR data terminal. DNOS returns an error code when the Read Direct operation is specified for a 733 ASR.

The extended call flag in the user flag field (byte 5) of the supervisor call block must be set to one for resource-specific I/O operations. Otherwise, the system does not use the extensions to the supervisor call block. The flags in the user flag field that apply to resource-specific I/O operations are:



2279509

Bit 1 — Reply flag. When the character validation flag is set to zero, set the reply flag to one for a Write with Reply, Remote Get Event Character, or Read with Validation operation, or set the reply flag to zero when a previously supplied validation table applies.

Bit 6 — Extended call flag. Set as follows:
1 — Extended call block (required for resource-specific I/O).
0 — Basic supervisor call block (used for resource-independent I/O).

The extension to the basic supervisor call block is as follows:

DEC	HEX		
12	C	VALIDATION TABLE/REPLY BLOCK ADDRESS	
14	E	EXTENDED USER FLAGS	
16	10	[RESERVED]	<EVENT BYTE>
18	12	[RESERVED]	
20	14	[RESERVED]	

2279510

The extension to the call block contains the following:

Byte **Contents**

12–13 Character validation table address (when character validation is specified in the extended user flags). The address of a table of character validation data. Reply block address (when the reply flag is set to one). The address of a block containing the address and count fields for a Write with Reply operation.

14–15 Extended user flags field. Contains sixteen flags that apply to all or some of the terminal operations as described in succeeding paragraphs.

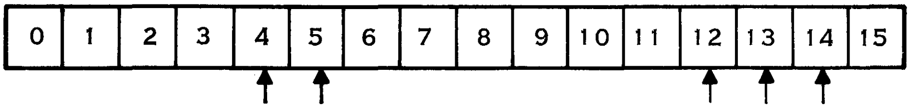
16 [Reserved]. 733 ASR data terminal I/O ignores any data in this field, which allows an extended call block to be used for either VDT or 733 ASR data terminal I/O.

17 <Event byte>. The system stores an event character in this field when the terminal has been opened in the event mode and an event key is pressed.

18–19 [Reserved]. 733 ASR data terminal I/O ignores any data in this field, which allows an extended call block to be used for either VDT or 733 ASR data terminal I/O.

20–21 [Reserved]. 733 ASR data terminal I/O ignores any data in this field, which allows an extended call block to be used for either VDT or 733 ASR data terminal I/O.

The extended user flags are:



2279511

The following lists the flags and the I/O operations in which they are effective. Detailed descriptions of the uses of the flags follow in subsequent paragraphs.

Bit	Definition	Used in Operations
4	Eight-bit ASCII	Read ASCII and Write ASCII
5	Task edit	Read
12	No echo	Read ASCII
13	Character validation	Read
14	Validation error mode	Read

6.4.3.1 Eight-Bit ASCII. This flag, when set to one, forces a line feed and carriage return at the end of a record during a Read ASCII or Write ASCII operation. This flag does not apply to terminals using JISCII.

6.4.3.2 Task Edit. This flag, when set to one, causes any of the task edit characters listed in Table 6-3 that are entered during an input operation to terminate the operation and to be returned in byte 17 (event byte field) of the extended call block. When the flag is set to zero, the device service routine (DSR) ignores these characters.

6.4.3.3 No Echo. This flag, when set to one, inhibits the printing of characters entered at the keyboard. When a key is pressed, a blank is printed. When the flag is set to zero, each character is printed as it is entered. This flag applies only to Read ASCII operations.

6.4.3.4 Character Validation. This flag, when set to one, enables character validation of the field being read by a Read ASCII operation. Character validation is discussed in greater detail in the character validation paragraph of this section. When the character validation flag is set to zero, no character validation is performed. Refer to the description of the Write ASCII operation for the use of the reply flag when the character validation flag is set to zero.

6.4.3.5 Validation Error Mode. The validation error mode flag, when set to one, enables correction of errors detected during validation of field contents by the task. The Validation Error Mode operation is effectively a Reread operation; the flags that apply to a Read apply in the same way to this operation. When the user reenters one or more characters in the field, the system sets the validation error mode flag to zero. When the calling task sets the validation error mode flag to zero, the operation is performed in the normal mode.

6.4.3.6 Read ASCII Example. When a terminal has been opened in the event key mode and task edit keys are also enabled (carriage control flag set to one), either an event key or a task edit key may terminate a Read operation. The task edit character is always returned in the event byte of the extended call block, and the event character is also returned in that byte in resource-specific I/O. The state of the event key flag in the system flag field indicates which type of character is in the event byte when both are enabled. The task accesses and decodes the character and performs the function corresponding to the key.

The following is an example of the code for a Read ASCII operation with event key termination enabled by the previous Open operation and task edit key termination enabled for the Read operation:

```
REHC     DATA 0                   READ FIELD OF TERMINAL AT LUNO >3F.
         BYTE 9,>3F               FIELD SIZE IS 15 CHARACTERS.
SYSFL    BYTE 0                   EVENT KEYS AND TASK EDIT KEYS
         BYTE >02                 ENABLED.
         DATA RBUFF
         DATA 15
         DATA 0
         DATA 0
         DATA >0400
         DATA 0
         DATA 0
         DATA 0
```

6.4.3.7 Character Validation Operation. A Read ASCII operation may specify character validation. The characters of the field being read by the operation that are not within the range or ranges applicable to the operation are not accepted. Characters that are within the range or ranges are stored in the read buffer specified for the operation. The character validation flag in the extended user flags field is set to one for a Read with validation operation.

Each Read operation with character validation must specify a validation table. Specifying a validation table requires:

- Setting the character validation flag to one
- Supplying a validation table
- Placing the address of the table in the character validation table address field of the extended call block

The validation table contains one or more ranges of characters that define the valid characters for the field. The table may define the valid characters by specifying ranges of characters that are not valid, or by specifying ranges of characters that are valid. Each range in the table requires two bytes, and the table contains two bytes of overhead. Thus the length of the table in bytes is two times the number of ranges, plus two. The format of the table is as follows:

DEC	HEX		
0	0	LENGTH	FLAGS
2	2	RANGE 1 LOW CHAR.	RANGE 1 HIGH CHAR.
2N	2N	RANGE n LOW CHAR.	RANGE n HIGH CHAR.

2279512

The validation table contains the following:

Byte	Contents
0	Length — Length of the validation table in bytes (2n + 2).
1	Flags: Bit 0 — Validation flag. Set as follows: 1 — Invalid ranges. Characters greater than or equal to the low character and less than or equal to the high character are invalid. 0 — Valid ranges. Characters greater than or equal to the low character and less than or equal to the high character are valid. Bits 1-7 — Reserved.
2	Low character for range 1.
3	High character for range 1.
	Character pairs for additional ranges.
2n	Low character for range n.
2n + 1	High character for range n.

Character validation is performed after each character is entered in the field.

The user must press one of the following correction keys when the last character entered is invalid:

- (CTRL) H or BACKSPACE
- RUB OUT
- (CTRL) N
- (CTRL) T

Next, the user enters the data correctly.

An example Read with Validation operation performs the following:

- Reads a 10-character field.
- Validates the field as an alphanumeric field with no lowercase letters.

The following is an example of the code for the supervisor call block and validation table for the example operation:

```
RVAL      DATA 0                  READ FIELD OF TERMINAL AT LUNO >2B,
          BYTE 9,>2B              VALIDATING PER TABL. FIELD
          BYTE 0                  SIZE IS 10 CHARACTERS. READ
FLG1      BYTE >02                BUFFER IS BUFF.
          DATA BUFF
          DATA 10
          DATA 0
          DATA TABL
FLG2      DATA >0004
          BSS 6
           .
           .
           .
          EVEN
TABL      BYTE 6                  LENGTH OF TABLE
          BYTE 0                  VALID RANGES
          DATA >3039              RANGE 1 — NUMERALS
          DATA >415A              RANGE 2 — UPPERCASE LETTERS
```

6.4.3.8 Field Validation. Any validation of a field must be performed by a task following the reading of the field. This could verify that the field contains the proper number of letters, followed by numbers, for example. The Read ASCII operation in the validation error mode is used by the task to obtain corrected data when an error has occurred. Character validation may be requested for the operation also. The validation error mode flag is set to one to enable the mode.

In the validation error mode, the operation requires reentry of the field. The correction key is the BACKSPACE key. When a character is entered prior to pressing the BACKSPACE key, the character is not printed (no echo).

The difference between a normal Read ASCII operation and one that specifies the validation error mode is that in the validation error mode input is ignored until the BACKSPACE key is pressed. The user should backspace to the leftmost error character and enter the correct characters in the remainder of the field.

The call block for the previous Read ASCII operation can be used by setting the validation error mode flag to one. The following instructions set the flag in the call block of the character validation coding example:

```
MASK2     BYTE  >2
          SOCB  @MASK2,@FLG2 + 1
```

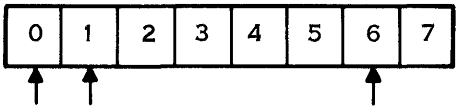
6.4.3.9 Getting Event Characters. The Remote Get Event Character operation (sub-opcode >05) returns an event character in the event byte (byte 17) of the extended supervisor call block. The LUNO assigned to the terminal does not have to be open. The operation is an alternative to performing a Read operation to obtain an event character.

The operation is a special type of Read Device Status operation; the number of characters stored in the input character queue is returned also.

The following fields of the extended supervisor call block apply to a Remote Get Event Character operation:

- SVC code — 0
- Return code
- Sub-opcode — >05
- Logical unit number (LUNO)
- User flags
- Data buffer address
- Read character count
- <Actual read count>
- Event byte

The following user flags apply to a Remote Get Event Character operation:



2279513

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

Bit 1 — Reply flag. Set to one.

Bit 6 — Extended call flag. Set to one.

The logical unit number (LUNO) field contains the LUNO assigned to the device at which the event character is entered.

The data buffer address is the address of the buffer into which DNOS places the status information.

The read character count is the length of the buffer.

DNOS returns the number of characters stored in the buffer in the actual read count field. The system returns four in this field when the specified LUNO is assigned to a 733 ASR data terminal.

DNOS returns the event character in the event byte.

After a Remote Get Event Character operation returns the status of a 733 ASR data terminal, the data buffer contains the following:

Byte	**Contents**
0-1	>FFFF.
2-3	Number of characters currently buffered in the input character queue.

DNOS maintains an input character queue that stores characters input while the system is processing a previously-entered character or command. Bytes 2 and 3 contain the number of characters in the queue. The size of this queue is specified when the system is generated.

The following is an example of the source code for a supervisor call block for a Remote Get Event Character operation and code for the read buffer:

```
RGEVCH   DATA 0                 GET EVENT CHARACTER FROM TERMINAL
         BYTE 5,>32             ASSIGNED TO LUNO >32.
         DATA >42
         DATA MRADR
         DATA 10
         DATA 0
         DATA 0
         DATA 0
EVCHAR   BYTE 0,0
         DATA 0,0
MRADR    BSS 1                  DEVICE STATUS BUFFER
         BSS 1
CHINQ    BSS 8
```

6.5 TELEPRINTER DEVICE I/O

DNOS supports both resource-independent (short SVC call block) and resource-specific (extended SVC call block) I/O for both Keyboard Send/Receive (KSR) and Receive Only (RO) teleprinter terminals (TPDs). These teleprinter devices are usually general purpose portable terminals.

Resource-independent I/O for the TPDs includes operations that are analogous to sequential file operations. Resource-specific I/O for TPDs includes operations that apply only to TPDs. These operations give a program control over functions of special keys. Except for the Read Device Status operation, the device must be opened using sub-opcode >00 or >03 prior to any I/O operation.

The subset of sub-opcodes for the TPDs applies to both resource-independent and resource-specific I/O, as follows:

00	Open
01	Close
02	Close, Write EOF
03	Open and Rewind
04	Close and Unload
05	Read Device Status and Get Remote Event
09	Read ASCII
0A	Read Direct
0B	Write ASCII
0C	Write Direct
0D	Write EOF
0E	Rewind
0F	Unload (Hang-up, Resource-Specific Only)
15	Set Device Characteristics (Resource-Specific Only)

The following I/O SVC block for TPD operations is the basic block used for all operations. If an extension to this block is necessary for a particular operation, it is indicated in the operation description.

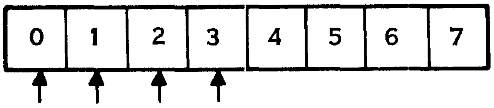
SVC >00 -- I/O OPERATIONS

ALIGN ON WORD BOUNDARY
CAN BE INITIATED AS AN EVENT

DEC	HEX		
0	0	>00	<RETURN CODE>
2	2	SUB-OPCODE	LUNO
4	4	<SYSTEM FLAGS>	USER FLAGS
6	6	DATA BUFFER ADDRESS	
8	8	READ CHARACTER COUNT	
10	A	WRITE CHARACTER COUNT/<ACTUAL READ COUNT>	

2279470

The system flags (byte 4) in the supervisor call block apply to all TPD I/O. These flags are:



2283187

Bit 0 — Busy flag. Set by system as follows:
1 — Busy.
0 — Operation completed.

Bit 1 — Error flag. Set by system as follows:
1 — Error.
0 — No error.

Bit 2 — End-of-file. Set by system as follows:
1 — (CTRL) Y key terminated the operation.
0 — Operation terminated without the (CTRL) Y key being pressed.

Bit 3 — Event key flag. Set by system as follows:
1 — An event key terminated the operation.
0 — Operation terminated without an event key being pressed.

The user flags (byte 5) in the supervisor call block apply to all TPD I/O. However, significance of these flags differs for various operations. The flags that apply to each operation are described in the detailed description of each operation.

The character set applicable to a TPD is the ASCII character set or the JISCII character set appropriate to the country code for which the system was generated.

6.5.1 Key Categories

The system interprets each key on the keyboard in one or more of the following categories:

- Data
- Hold
- Event
- System edit
- Task edit

The Device Character Set appendix to this manual lists the keys on TPD terminal keyboards and the type of each key.

6.5.1.1 Data Keys. The data keys return the codes of printable characters to the buffer specified in the call block. The category includes the keys that return ASCII codes >20 through >7E.

6.5.1.2 Hold Key. The hold key suspends output to the TPD. Operation may be resumed by pressing any other key except RETURN, exclamation point (!), or (CTRL) X. The hold key is the (CTRL) S key. The first key pressed after the hold key is not placed in the read buffer.

When the RETURN key is pressed following the hold key, the current operation is aborted, and an error code is returned to the task that requested the I/O.

When the exclamation point key is pressed following the hold key, the system activates SCI and the output continues. Contention between the interrupted output and SCI for the use of the TPD may cause unpredictable results.

When (CTRL) X is pressed following the hold key, a hard break results. The hard break terminates the current task and activates SCI. The end action specified by the task, if any, is performed prior to terminating the task and activating SCI. The hard break should be used to abort tasks when appropriate; it should be used with care because the hard break aborts pending I/O requests.

The system selects the task to be terminated by the hard break from among the tasks of the current job. The following rules apply in the listed order of priority:

1. When only the System Command Interpreter (SCI) task is active, SCI is terminated.
2. Any other foreground task is terminated.
3. Any other background task is terminated.
4. When other tasks are active along with SCI, SCI is terminated last.

There may be times when the hard break terminates a task other than the one intended. In that case, press the hold key and (CTRL) X again to terminate the intended task.

The effect of a subsequent hard break depends upon the timing. When the task has not yet taken end action in response to the first hard break, the subsequent hard break terminates the next task in the order of priority. When the task is executing the end action routine but has not completed end action, the task is aborted as if end action had not been provided. When end action has completed, the subsequent hard break causes the task to take end action again.

The end action routine should execute a Get End Action Status SVC to obtain the error code and identify the cause of the termination. When the task error code returned by the SVC is >10, a hard break has occurred. The task may process the hard break and resume execution after executing a Reset End Action Status SVC. The task must place the WP, PC, and ST values returned by the Get End Action Status SVC in R13, R14, and R15 and execute an RTWP instruction to resume execution.

6.5.1.3 Event Keys. Activating the event key mode enables use of event keys as task programmable function keys. Opening the LUNO assigned to the TPD, with the event key mode flag (bit seven in the user flags) set to one, activates the event key mode. A Read ASCII operation, using the extended call block with the task edit flag set, returns the event character to the task. The Read Operation does not return event keys to the task or remove event characters from the buffer when the device is opened with a regular (not extended) call block. Event characters can be accessed by a Remote Get Event Character operation without opening the LUNO assigned to the TPD.

When an event key is pressed while using an extended call block, the corresponding character code is stored in the event character buffer. When an input operation using an extended call block is being performed, the operation terminates with the event key bit (bit 3 of byte 4 of the call block) set to one.

When no input operation is being performed and an event key is pressed, the next input operation is immediately terminated with the event key bit of the call block set to one.

The Device Character Set appendix to this manual describes the event keys. The actual key pressed to generate a code can vary with different terminals. Refer to the specific manual for the terminal.

The event keys for the 74x, 76x, 78x TPDs are:

(CTRL) 3   (CTRL) A   (CTRL) B
(CTRL) C   (CTRL) D   (CTRL) E
(CTRL) F   (CTRL) V   (CTRL) W
(CTRL) X   (CTRL) Y   (CTRL) Z
(CTRL) ,   (CTRL) +   (CTRL) .
(CTRL) /   (CTRL) [

The event keys for the 820 TPD are:

(CTRL) 3   (CTRL) A   (CTRL) B
(CTRL) C   (CTRL) D   (CTRL) E
(CTRL) F   (CTRL) V   (CTRL) W
(CTRL) X   (CTRL) Y   (CTRL) Z
(CTRL) \   (CTRL) {   (CTRL) =
(CTRL) -

The task decodes the event character and performs the desired function. When the input operation that terminates with the event key bit set to one uses the extended call block, (resource-specific I/O) the event character is returned in the event character field. The event character can be obtained without performing a Read operation (and without opening the LUNO) by performing a Remote Get Event Character operation. When the input operation uses the basic I/O call block (resource-independent I/O), the following occurs:

1. The event character is not returned because there is no event byte.

2. Nothing is placed in the read buffer.

3. The character is not removed from the character queue.

4. The count of characters in the character queue is not decremented.

This means the next input operation will also terminate with the event bit set.

NOTE

> Any task may access the event character buffer by performing I/O to any LUNO assigned to the TPD. The first access is the only access that returns the correct character. Tasks that perform I/O to a TPD to which SCI is performing I/O must avoid accessing event characters. Either the task or SCI may fail to perform the intended function.

6.5.1.4 System and Task Edit Keys. System edit keys are control keys that are implemented by the system. Task edit keys are control keys that are task functions. Five of the keys are both system and task edit keys, for which the system performs a function. The task may perform an additional function.

Task edit functions apply only to read ASCII operations using the extended call block. The task edit flag (byte 14, bit 5) in the extended user flags must be set to one to enable task edit functions. The device service routine (DSR) returns the character code of the task edit character in the event byte field of the extended call block.

6.5.2 TPD Terminal Resource-Independent I/O

The operations appropriate for TPDs are described in subsequent paragraphs. The following sub-opcodes do not apply to TPDs and produce the indicated results:

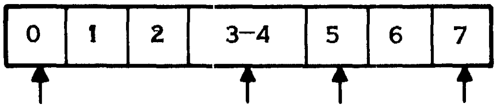
Sub-opcode	Operation	Action
06	Forward Space	Ignored
07	Backward Space	Ignored

6.5.2.1 Open. Sub-opcode >00 specifies an Open operation. The Open operation causes the TPD to perform a line feed and a carriage return and is required for TPDs. DNOS does not automatically validate the Open operation unless the device was specified with VALIDATE OPENS? YES during system generation; that is, a possible conflict with I/O to the same device by another task is not detected. An Open operation is not required prior to performing a Read Device Status operation.

The following fields of the basic supervisor call block apply to an Open operation:

- SVC code — 0
- Return code
- Sub-opcode — >00
- Logical unit number (LUNO)
- User flags
- Data buffer address
- Read character count

The following user flags apply to an Open operation:



2283188

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

Bits 3-4 — Access privilege flag. Set as follows:
00 — Exclusive write.
01 — Exclusive all.
10 — Shared.
11 — Read only.

Bit 5 — Immediate open flag. Set as follows:
1 — System immediately returns control to the calling task.
0 — System verifies that the device is connected to the port before returning control to the calling task.

Bit 7 — Event key mode flag. Set as follows:
1 — Enable event key mode.
0 — Disable event key mode.

If bit 5 of the user flags is set to one, the call completes immediately regardless of whether the port is connected or unconnected. Otherwise, the call completes after the port is connected.

The logical unit number (LUNO) field contains the LUNO assigned to the TPD to be opened.

The Open operation returns the device type code in the data buffer address field (bytes 6 and 7) of the supervisor call block. The device type code for a TPD is >0001.

When the calling task places zero in the read character count field (bytes 8 and 9) of the supervisor call block, the Open operation returns the default logical record length for the device. The default logical record length for a TPD is >56.

A TPD must be opened with the event key flag set to one if event keys are to be used as task programmable function keys.

To access an event key character, perform a Remote Get Event Character operation.

The following is an example of the source code for a supervisor call block to open a TPD:

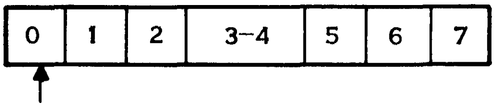
```
OHCT     DATA 0                        OPEN TERMINAL ASSIGNED TO LUNO >20.
         BYTE 0,>20
         DATA 0
TPE      DATA 0
LRL      DATA 0
         DATA 0
```

6.5.2.2 Close. Sub-opcode >01 specifies a Close operation. The Close operation ends I/O to a LUNO from the calling task. The LUNO remains assigned to the device, and may be opened again for additional I/O operations. DNOS writes a carriage return character to the device to which the LUNO is assigned. When a task terminates, DNOS closes all LUNOs that the task opened.

The following fields of the basic supervisor call block apply to a Close operation:

- SVC code — 0
- Return code
- Sub-opcode — >01
- Logical unit number (LUNO)
- User flags

The following user flag applies to a Close operation:



2283189

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

The logical unit number (LUNO) field contains the LUNO to be closed.

The following is an example of the source code for a supervisor call block to close a TPD:

```
CHCT      DATA 0                   CLOSE TERMINAL ASSIGNED TO LUNO >20.
          BYTE 1,>20
          DATA 0
          DATA 0
          DATA 0
          DATA 0
```

6.5.2.3 Close, Write EOF. The Close, Write EOF operation, sub-opcode >02, performs three line feed operations on Silent 700 series devices or a form feed on 800 series printers, followed by a Close operation.

6.5.2.4 Open and Rewind. The Open and Rewind operation, sub-opcode >03, is an Open operation followed by a Rewind operation. For TPDs, the Rewind operation consists of clearing the input character queue. The Open and Rewind operation causes the terminal to perform a page eject. If the attached TPD does not support page eject, three line feeds are performed. If the Open and Rewind operation uses an extended call block with byte 14, bit 4 set to one, the input queue is flushed and the device is placed in the eight-bit ASCII mode. Eight-bit ASCII is intended for communicating with special, nonstandard devices. In the eight-bit ASCII mode, all characters received are passed directly to the calling task without checking for special characters or parity. When the device is in this mode the calling task must assume responsibility for handling special characters (hold, abort, etc.), line turn-around characters if the line is half-duplex, and checking parity. The device in this mode cannot bid SCI. An Open Rewind call block with byte 14, bit 4 set to zero restores the device to normal operating mode. The Open Rewind operation is the recommended way to set or reset the eight-bit ASCII mode since it ensures the characters passed to the calling task are the specified type by clearing the character queue.

6.5.2.5 Close and Unload. The Close and Unload operation, sub-opcode >04, performs the same function as the Close and drops the communications line.

6.5.2.6 Read Device Status. Sub-opcode >05 specifies a Read Device Status operation. The Read Device Status operation returns into the data buffer as much of the device information block as specified (up to a maximum of 64 characters) by the user in the read character count field.

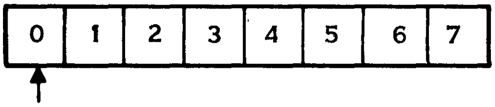
The following fields of the basic supervisor call block apply to a Read Device Status operation:

- SVC code — 0
- Return code
- Sub-opcode — >05
- Logical unit number (LUNO)
- User flags
- Data buffer address

- Read character count

- <Actual read count>

The following user flag applies to a Read Device Status operation:



2283190

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

The logical unit number (LUNO) field contains the LUNO assigned to the device for which status information is returned.

The data buffer address is the address of the buffer into which DNOS places the status information.

The read character count is the number of characters of device information desired. The read buffer must be large enough to hold the information.

DNOS returns the number of characters stored in the buffer in the actual read count field. The number of characters cannot exceed 64.

The contents of the data buffer after a Read Device Status operation has returned the status of a TPD is in Table 6-4.

Table 6-4. Status of Teleprinter Devices

Byte	Contents
0–1	>FFFF
2–3	Number of characters buffered in input character queue
4	DSR type: 5 = TPD
5	Reserved
6–7	Hardware interface CRU/TILINE address
8–9	Associated ACU hardware interface CRU address or >FFFF if no CRU is defined
10	ISR type: 1 = CI401 5 = TTY/EIA 6 = Any 9902 port
11	Reserved
12–13	Read ASCII time-out in ¼ seconds
14–15	Write ASCII and direct time-out in ¼ seconds
16–17	Read direct time-out in ¼ seconds
18–19	Read direct time-out for characters 2-N
20	Reserved
21	State flags
22	Line flags: 0 = half-duplex 1 = switched line 2,7 = as currently defined
23	Access flags
24	Speed code: – 1 = modem selected 0 = 110 baud 2 = 300 baud 3 = 1200 baud 4 = 2400 baud 5 = 4800 baud 6 = 9600 baud
25	EOR character
26	EOF character
27	LTA character
28	Parity error substitute character
29	Carriage return delay count
30–33	Reserved
34–35	Count of maximum characters buffered

Table 6-4. Status of Teleprinter Devices (Continued)

Byte	Contents
36	Terminal type: >03 = 703 >07 = 707 >2B = 743 >2D = 745 >3F = 763 >41 = 765 >51 = 781 >53 = 783 >55 = 785 >57 = 787 >58 = 820 >7D = 825 >8C = 840
37	Last character received
38-39	Saved extended flags
40	Reserved
41	Sysgen speed
42-43	Reserved
44-45	Sysgen time-out value in 250-ms increments
46-47	Number of parity errors
48-49	Number of lost characters
50-51	Number of reads
52-53	Number of writes
54-55	Number of other I/O calls
56	Number of retries
57	Number of LUNOs assigned
58-59	Number of read errors
60-61	Number of write errors
62-63	Number of other I/O errors

DNOS maintains an input character queue that stores characters input while the system is processing a previously-entered character or command. Bytes 2 and 3 contain the number of characters in the queue. The maximum size of this queue is specified when the system is generated.

The following is an example of the source code for a supervisor call block for a Read Device Status operation and the read buffer. This call block returns the first 10 characters of the device information block (through the ACU CRU address). The number of characters currently buffered in the input queue is returned as part of this status information at the label LIQ.

```
RDSHCT  DATA 0          READ STATUS OF TERMINAL ASSIGNED TO
        BYTE 5,>35      LUNO >35.
        DATA 0
        DATA DMY
        DATA 10
        DATA 0
DMY     BSS 2           DEVICE STATUS BUFFER
LIQ     BSS 8
```

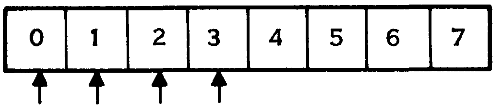
6.5.2.7 Read ASCII. Sub-opcode >09 specifies a Read ASCII operation. The Read ASCII operation reads a record from the keyboard and stores the characters in the specified buffer, two characters per word.

If the number of characters received from the device is greater than the length of the read buffer, the excess characters are stored in the input queue and returned by the next Read Direct or Read ASCII operation. Additional characters, received after the queue is full, are discarded and the next operation that is performed by the DSR terminates with an error (KSB queue overflow). System generation specifies the size of the KSB input queue.

The following fields of the basic supervisor call block apply to a Read ASCII operation:

- SVC code — 0
- Return code
- Sub-opcode — >09
- Logical unit number (LUNO)
- <System flags>
- User flags
- Data buffer address
- Read character count
- <Actual read count>

The following system flags apply to a Read ASCII operation:



2283187

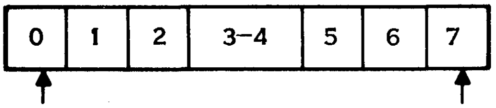
Bit 0 — Busy flag. Set by system as follows:
1 — Busy.
0 — Operation completed.

Bit 1 — Error flag. Set by system as follows:
1 — Error.
0 — No error.

Bit 2 — End-of-file. Set by system as follows:
1 — (CTRL) Y key terminated the operation.
0 — Operation terminated without the (CTRL) Y key being pressed.

Bit 3 — Event key flag. Set by system as follows:
1 — An event key terminated the operation.
0 — Operation terminated without an event key being pressed.

The following user flags apply to a Read ASCII operation:



2283192

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

Bit 7 — Blank adjustment flag. Set as follows:
1 — Read with blank adjustment.
0 — Read without blank adjustment.

The logical unit number (LUNO) field contains the LUNO assigned to the TPD from which a record is to be read.

The data buffer address is the address of the buffer into which DNOS places the record.

The read character count is the length of the buffer.

DNOS returns the number of characters stored in the buffer in the actual read count field.

The Read ASCII operation recognizes the characters listed in the Device Character Set appendix for TPDs. When the operation stores the characters, it packs them one per byte. When the country code in effect is not >0200 (Japan), the most significant bit is set to zero. When the country code is >0200, the eight-bit JISCII code is stored. The operation continues until one of the following occurs:

- The RETURN key ((CTRL)M on some terminals) is pressed or the DSR receives a >0D character
- The buffer is full
- An event key is pressed (if the TPD terminal is in the event key mode)
- The (CTRL)Y is pressed or the DSR receives a >99 or >19 character
- The time-out period elapses (if one is specified during system generation)
- The DSR encounters an error

Characters can be corrected by pressing the (CTRL) H or BACKSPACE key. The TPD performs a backspace operation and deletes the previously entered character from the data buffer each time the key is pressed. The first time the key is pressed, the printer also performs a line feed operation. After spacing to the character in error, reenter the characters deleted.

When the RETURN key is pressed, the number of characters entered is stored in the actual read count field and the operation terminates.

On Read ASCII operations the following edit functions are performed internally: backspace (BS), erase field, tab, line feed (LF), carriage return (CR), end-of-record, and end-of-file (EOF). The corresponding keycodes are not placed in the read buffer.

When the TPD is opened in the event key mode and an event key is pressed, the system sets the event key flag in the system flags byte and terminates the operation. The event character is returned in the event byte of the call block if an extended call block is used. The event character can also be accessed by performing a Remote Get Event Character operation.

When an event key is pressed between Read operations, the next Read operation performed after pressing the event key terminates with the event key flag set and zero in the actual read count field.

When the (CTRL) Y key is pressed, the system sets the EOF flag in the system flags byte and terminates the operation.

When blank adjustment is specified for variable length records, blanks are stored in the buffer to fill the record. That is, when the record length is less than the buffer size, the device service routine (DSR) supplies blanks (>20) to fill the buffer. The character count returned in bytes 10 and 11 includes the blanks supplied by the DSR.

The following is an example of the source code for a supervisor call block for a Read ASCII operation and code for the read buffer:

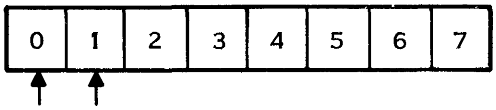
```
RDHCT    DATA 0              READ RECORD FROM TERMINAL ASSIGNED
         BYTE 9,>2C          TO LUNO >2C IN THE INITIATE I/O
         BYTE 0,>80          MODE.
         DATA RBUF
         DATA 80
         DATA 0
RBUF     BSS 80              READ BUFFER
```

6.5.2.8 Write ASCII. Sub-opcode >0B specifies a Write ASCII operation. The Write ASCII operation transfers a record from the specified buffer to the TPD. DNOS also supports an optional Write with Reply operation, which is effectively a Write operation followed by a Read ASCII operation.

The following fields of the basic supervisor call block apply to a Write ASCII operation:

- SVC code — 0
- Return code
- Sub-opcode — >0B
- Logical unit number (LUNO)
- <System flags>
- User flags
- Data buffer address
- Write character count
- Reply block address (Write with Reply operation)

The following system flags apply to a Write ASCII operation:

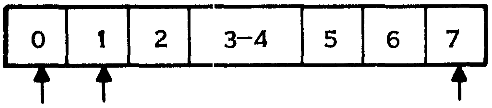


2283151

Bit 0 — Busy flag. Set by system as follows:
1 — Busy.
0 — Operation completed.

Bit 1 — Error flag. Set by system as follows:
1 — Error.
0 — No error.

The following user flags apply to a Write ASCII operation:



2283193

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

Bit 1 — Reply flag. Set as follows:
1 — Write operation followed by a Read operation.
0 — All other operations.

Bit 7 — Blank adjustment flag. Set as follows:
1 — Write with blank adjustment.
0 — Write without blank adjustment.

The logical unit number (LUNO) field contains the LUNO assigned to the TPD to which a record is to be written.

The data buffer address is the address of the buffer that contains the record to be printed.

The write character count is the number of characters to be printed by the TPD.

The Write ASCII operation prints a record on the TPD. The record consists of ASCII characters or JISCII characters, as specified by the country code.

The ASCII characters from >00 through >1F are control and protocol characters and should be avoided in normal data traffic. The actions taken by a TPD depend on the specific terminal and its configuration; refer to the manual for the terminal being used. When these characters are transmitted to the TPD these results occur:

- ENQ (>05) — The TPD transmits the answer back memory if it is installed.
- BEL (>07) — The bell rings.
- HT (>09) — The TPD prints a space.
- LF (>0A) — A line feed occurs.
- Form feed (>0C) — The TPD attempts a form feed if it is capable of performing a form feed.
- CR (>0D) — The TPD performs a carriage return.
- ESC (>1B) — Some TPDs interpret this as the beginning of an Extended Device Control (EDC) sequence and attempt to interpret the following characters as meaningful terminal orders; the results can be unexpected.

When blank adjustment is specified, trailing blanks in the buffer are not written. The write character count in bytes 10 and 11 is not altered.

A Write with Reply operation requires the following in addition to the requirements for a Write ASCII operation:

- The reply flag in the user flags byte set to one
- The extension to the supervisor call block
- The reply block

The extension to the basic I/O supervisor call block is as follows:

DEC	HEX	
12	C	REPLY BLOCK ADDRESS

2283194

The reply block is a three-word block, containing addresses for the Read operation, as follows:

DEC	HEX	
0	0	DATA BUFFER ADDRESS
2	2	READ CHARACTER COUNT
4	4	<ACTUAL READ COUNT>

2283195

The three fields are identical to the corresponding fields of the supervisor call block for a Read ASCII operation.

The following is an example of the source code for a supervisor call block for a Write ASCII operation:

```
WAHCT   DATA 0            WRITE RECORD TO TERMINAL ASSIGNED TO
        BYTE >B,>4C       LUNO >4C INITIATE MODE.
        BYTE 0,>80
        DATA WRBUFF
        DATA 0
        DATA 80
```

The following is an example of the source code for a supervisor call block for a Write ASCII operation that uses the optional Write with Reply operation:

```
WRHCT     DATA 0                       WRITE RECORD TO TERMINAL ASSIGNED TO
          BYTE >B,>4C                  LUNO >4C INITIATE MODE AND
          BYTE 0,>C0                   WRITE WITH REPLY.
          DATA WRBUFF
          DATA 0
          DATA 80
          DATA RBK
```

The reply block is coded as follows:

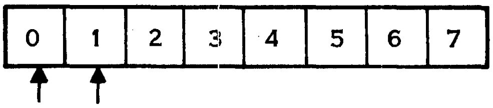
```
RBK       DATA REPLY                   REPLY BUFFER ADDRESS
          DATA 80                      MAXIMUM LENGTH OF REPLY
          DATA 0                       REPLY CHARACTER COUNT
```

6.5.2.9 Write EOF. The Write EOF operation (sub-opcode >0D) performs a page eject on TPDs that support the page eject operation or three line feed operations on TPDs that do not support page eject.

The following fields of the basic supervisor call block apply to a Write EOF operation:

- SVC code — 0
- Return code
- Sub-opcode — >0D
- Logical unit number (LUNO)
- <System flags>
- User flags

The following system flags apply to a Write EOF operation:

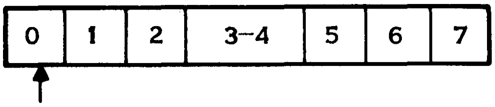


2283196

Bit 0 — Busy flag. Set by system as follows:
1 — Busy.
0 — Operation completed.

Bit 1 — Error flag. Set by system as follows:
1 — Error.
0 — No error.

The following user flags apply to a Write EOF operation:



2283189

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

The logical unit number (LUNO) field contains the LUNO assigned to the TPD to which a record is to be written.

The following is an example of the source code for a supervisor call block for a Write EOF operation:

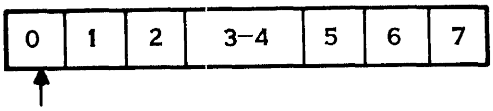
```
WEHCT    DATA 0              WRITE EOF TO TERMINAL ASSIGNED TO
         BYTE >D,>4C         LUNO >4C INITIATE MODE.
         BYTE 0,>80
         DATA 0
         DATA 0
         DATA 0
```

6.5.2.10 Rewind. Sub-opcode >0E specifies a Rewind operation. The Rewind operation clears the input character queue and performs a page eject on TPDs that support the page eject operation or it performs three line feed operations on TPDs that do not support page eject. If the extended call block is used and bit 4 of the extended flags is set to one, the Rewind operation is performed and the device is placed in the eight-bit ASCII mode. If bit 4 is not set or the extended call block is not used, the rewind is performed and the device is restored to normal mode.

The following fields of the basic supervisor call block apply to a Rewind operation:

- SVC code — 0
- Return code
- Sub-opcode — >0E
- Logical unit number (LUNO)
- User flags

The following user flag applies to a Rewind operation:



2283189

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

The logical unit number (LUNO) field contains the LUNO assigned to the TPD that is to receive the Rewind operation.

The following is an example of the source code for a supervisor call block to rewind a TPD:

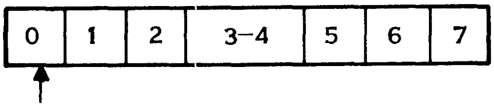
```
RWND    DATA 0              REWIND TPD ASSIGNED TO LUNO >4A.
        BYTE >E,>4A
        DATA 0
        DATA 0
        DATA 0
        DATA 0
```

6.5.2.11 Unload. Sub-opcode >0F specifies an Unload operation. The Unload operation disconnects a switched line that is attached to a TPD.

The following fields of the basic supervisor call block apply to an Unload operation:

- SVC code — 0
- Return code
- Sub-opcode — >0F
- Logical unit number (LUNO)
- User flags

The following user flag applies to an Unload operation:



2283189

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

The logical unit number (LUNO) field contains the LUNO assigned to the TPD that is to receive the Unload operation.

The following is an example of the source code for a supervisor call block to unload a TPD:

```
ULCS      DATA 0                     UNLOAD TPD ASSIGNED TO
          BYTE >F,>4B                LUNO >4B
          DATA 0
          DATA 0
          DATA 0
          DATA 0
```

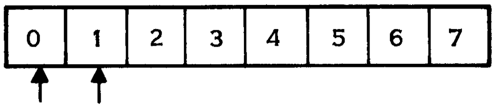
6.5.2.12 Device Dependent Communication Control. DNOS enables a task to access device dependent communications control through sub-opcode >15. The data buffer contains parameters and other sub-opcodes that specify operations available with device dependent communication control. The List Hardcopy Port Characteristics (LHPC) SCI command can display the state of most of these parameters. The Modify Hardcopy Port Characteristics (MHPC), Call Terminal (CALL), Answer Incoming Call (ANS), or Terminal Disconnection (DISC) SCI commands can modify most of these characteristics.

The following fields of the basic supervisor call block apply to a device dependent communication control.

- SVC code — 0
- Return code
- Sub-opcode — >15
- Logical unit number (LUNO)
- <System flags>
- User flags
- Data buffer address
- Write character count

The logical unit number (LUNO) field contains the LUNO assigned to the TPD.

The following system flags apply to device dependent communication control:



2283196

Bit 0 — Busy flag. Set by system as follows:
1 — Busy.
0 — Operation completed.

Bit 1 — Error flag. Set by system as follows:
1 — Error.
0 — No error.

The data buffer address is the address of the buffer that contains the sub-opcodes and parameters used in device dependent communication control. Each sub-opcode designates an operation of device dependent communication control. The parameters specify different functions and values of an operation.

The format of the data buffer is as follows:

BYTE		
0	SUB-OPCODE	>00
2	PARAMETER 1	PARAMETER 2
	≈	≈
2n	PARAMETER n-1	PARAMETER n

2283202

The data buffer contains the following:

Byte	Contents	
0	**Sub-opcode**	**Operation**
	>16	Modify Timing Characteristics
	>17	Modify Line Characteristics
	>18	Modify Terminal Type
	>19	Modify Special Characters
	>1A	Connect
	>1B	Clear Character Queue
	>1C	Set File Transfer Parameters
	>1D	Set Exclusive Access
	>1E	Set Shared Access
1	>00	
2-n	Parameters	

Each parameter is described for the operation to which it applies in the following paragraphs. All parameters are entered in hexidecimal.

The write character count is the number of characters in the data buffer.

Modify Timing Characteristics. Sub-opcode >16 specifies the Modify Timing Characteristics operation. This operation allows the default time-outs for a device to be changed. New values for time-outs are specified in the parameters of the data buffer as follows:

Byte	Time-Out Value (250-ms Increments)
2–3	Read Time-Out
4–5	Write Time-Out
6–7	Primary Read Direct Time-Out
8–9	Secondary Read Direct Time-Out

Modify Line Characteristics. Sub-opcode >17 specifies the Modify Line Characteristics operation. This operation modifies the configuration of a communication line with the following parameters in the data buffer:

Byte	Line Characteristics
2	LTA character. Set as follows: New LTA character. 0 — Do not change current LTA character.
3	Speed nn (where nn is value):

Value	Speed (asynchronous bps)
0	110
1	300
2	600
3	1200
4	2400
5	4800
6	9600
−1	300 or 1200 depending on the state of the interface-board. This enables automatic speed selection in conjunction with VA3400 and 212A modems. Not valid for configurations using direct-connect, half-duplex, or TTY/EIA interface.

Byte	Line Characteristics
4	Bit 0 — Half-duplex = 1 Bit 1 — Switched Bit 2 — Disabled Bit 3 — Auto-disconnect enabled Bit 4 — Require DLE and EOT for auto-disconnect Bit 5 — SCF ready/busy monitor Bit 6 — Exclusive access Bit 7 — LTA enable (half-duplex only)

Modify Terminal Type. Sub-opcode >18 specifies the Modify Terminal Type operation. This operation allows parameters related to a TPD type to be altered.

Byte	Function	
2	**Type Value**	**Terminal Model**
	>03	703
	>07	707
	>2B	743
	>2D	745
	>3F	763
	>41	765
	>51	781
	>53	783
	>55	785
	>57	787
	>58	820
	>7D	825
	>8C	840
3	Bit 0 — Echo. Set as follows: 0 — Echo to the TPD. 1 — Suppress echo to the TPD.	

Modify Special Characters. Sub-opcode >19 specifies the Modify Special Characters operation. This operation modifies the characters that indicate end-of-record and end-of-file. Specify new values for these parameters in the data buffer as follows:

Byte	Function
2	End-of-record. Set as follows: New end-of-record character. 00 — Do not change current end-of-record character.
3	End-of-file. Set as follows: New end-of-file character. 00 — Do not change current end-of-file character.

Connect. Sub-opcode >1A specifies the Connect operation. This operation establishes a connection to a TPD. If the initiate flag (bit 0) of the user flags is set to one, the task is not suspended. The task is suspended until the connection is complete if the initiate flag is set to zero. The values of the following parameters determine the method by which the connection to the TPD is made.

Byte	Value	Function
2	Nonzero	Establish request to send (RTS).
	Zero	Do not establish RTS.
3	Nonzero	Establish data terminal ready (DTR).
	Zero	Do not establish DTR.
4-5	Nonzero	Specify a time-out value in 250-ms increments.
	Zero	Zero specifies an infinite time-out.

If Ring Indicator or Data Set Ready is detected, the time-out reverts to 10 seconds for the duration of the connection. Thus, if a communications port is set to answer incoming calls with an infinite time-out and a device (not a modem) calls, the DSR terminates the call in 10 seconds if no connection is made. For full-duplex circuits, Data Carrier Detect must be sensed for the call to complete successfully.

Set File Transfer Parameters. Sub-opcode >1C specifies a Set File Transfer Parameters operation. This operation performs the following:

- Enables selection of a parity checking mode
- Selects time-outs
- Selects a parity error substitute character
- Disables the DC3-driven functions:
  - Bid
  - Hold output
  - Abort task
  - Time-out

The parameters of the Set File Transfer Parameters operation are defined and stored in the data buffer as follows:

Byte	Function
2–3	Primary time-out for Read Direct
4–5	Secondary time-out for Read Direct
6	Parity error substitute character
7	Bit 0 — Echo. Set as follows: 1— Suppress echo. 0— Enable echo. Bit 1 — Unused Bit 2 — Transmit parity. Set as follows: 1 — Enable transmit parity. 0 — Disable transmit parity. Bits 3–4 — Transmit parity type. Set as follows: 00 = Even. 01 = Odd. 10 = Mark. 11 = Space. Bit 5 — Receive parity. Set as follows: 1 — Enable receive parity. 0 — Disable receive parity. Bits 6–7 — Receive parity type. Set as follows: 00 = Even. 01 = Odd. 10 = Mark. 11 = Space.

The values of the parameters disappear when the terminal is disconnected.

Clear Character Queue. Sub-opcode >1B specifies the Clear Character Queue operation. This operation clears the input character queue. Depending on the state of the eight-bit ASCII flag in the extended flags word of the call block, this operation also sets the device into the eight-bit ASCII character mode or restores it to the normal operating mode. If the eight-bit ASCII flag is set to one, the device is placed in the eight-bit mode. If the flag is zero or a standard call block is used, the device is restored to the normal operating mode.

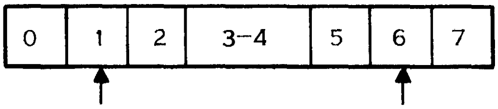
Set Exclusive Access. Sub-opcode >1D specifies the Set Exclusive Access operation. This operation places the communications port under control of the file transfer tasks. These tasks have bit 5 of the user flags set to one on Open operations.

Set Shared Access. Sub-opcode >1E specifies the Set Shared Access operation. This operation releases the communications port to tasks that do not have bit 5 of the user flags set to one on Open operations.

6.5.3 Teleprinter Device Resource-Specific I/O

Most of the resource-specific I/O operations use an eight-byte extension to the supervisor call block. The sub-opcodes for the resource-independent operations apply, but the operations are modified by the states of flags in the extended user flags field.

The extended call flag in the user flag field (byte 5) of the supervisor call block must be set to one for resource-specific I/O operations. Otherwise, the system does not use the extensions to the supervisor call block. The flags in the user flag field that apply to resource-specific I/O operations are:



2283203

Bit 1 — Reply flag. When the character validation flag is set to zero, set the reply flag to one for a Write with Reply or Remote Get Event Character.

Bit 6 — Extended call flag. Set as follows:
1 — Extended call block (required for resource-specific I/O).
0 — Basic supervisor call block (used for resource-independent I/O).

The extension to the basic supervisor call block is as follows:

Dec	Hex		
12	C	Validation Table/Reply Block Address	
14	E	Extended User Flags	
16	10	[Reserved]	<Event Byte>
18	12	[Reserved]	
20	14	[Reserved]	

2279510

The extension to the call block contains the following:

Byte	Contents
12–13	Character validation table address (when character validation is specified in the extended user flags). The address of a table of character validation data. Reply block address (when the reply flag is set to one). The address of a block containing the address and count fields for a Write with Reply operation.
14–15	Extended user flags field. Contains sixteen flags thatapply to all or some of the TPD operations as described in succeeding paragraphs.

Byte **Contents**

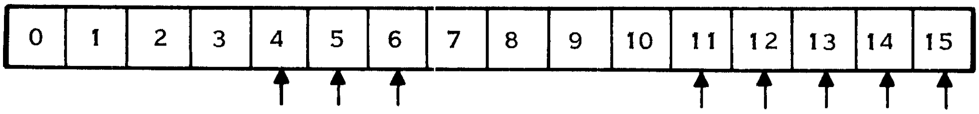
16 [Reserved]. TPD I/O ignores any data in this field, which allows an extended call block to be used for either VDT or TPD I/O.

17 <Event byte>. The system stores an event character in this field when the TPD has been opened in the event mode and an event key is pressed. During Read Direct eight-bit ASCII or Write Direct eight-bit ASCII with Reply operations, if bit 5 (Task Edit) of the extended flags is set to one, this field contains a user-specified termination character. The Read Direct eight-bit ASCII operation terminates if a character matching the character in this byte is encountered before the read buffer fills or before the time-out elapses (if it is specified). The terminating character is placed in the read buffer.

18–19 [Reserved]. TPD I/O ignores any data in this field, which allows an extended call block to be used for either VDT or TPD I/O.

20–21 [Reserved]. TPD I/O ignores any data in this field, which allows an extended call block to be used for either VDT or TPD I/O.

The extended user flags are:



2283152

The following lists the flags and the I/O operations in which they are effective. Detailed descriptions of the uses of the flags follow in subsequent paragraphs.

Bit	Definition	Used in Operations
4	Eight-Bit ASCII	Read Direct and Write Direct
	Append LF/CR/LTA	Read/Write ASCII
5	Task Edit	Read ASCII
	User specified with termination character	Read Direct eight-bit ASCII option
6	Beep	Read/Write ASCII
11	Forced Termination Character	Read ASCII
12	Echo	Read ASCII and Read Direct
13	Character Validation	Read ASCII
14	Validation Error Mode	Read ASCII
15	Warning Beep	Read ASCII

6.5.3.1 Eight-Bit ASCII or LF/CR/LTA. When this flag is set to one, DNOS supports eight-bit ASCII data on Read Direct and Write Direct operations. If this bit is set to one, DNOS forces a line feed (LF) and carriage return (CR) at the TPD after the end of a record during a Read or Write ASCII operation. If the current line turnaround (LTA) character is CR, an additional CR is not output at the TPD. If the current LTA is not CR, an additional CR is output. LTA only applies to a TPD if LTA is specified. This flag does not apply to TPDs using JISCII.

6.5.3.2 Task Edit. This flag, when set to one on a Read ASCII operation, causes the operation to terminate if any of the following types of characters are entered: task edit, event, system edit, or end-of-file. These characters are listed in the Device Character Set Appendix to this manual. The terminating character is returned in byte 17 (event byte field) of the extended call block. This flag, when set to one on a Read Direct eight-bit ASCII operation, causes the operation to terminate if a character is received that matches a character placed in byte 17 of the extended call block. The termination character is placed in the read buffer.

6.5.3.3 Beep. This flag, when set to one on Read ASCII operations, causes a BEL character to be sent to the TPD at the beginning of the operation. The TPD then sounds an audible tone to request the first input character. When the flag is set to one on Write ASCII operations, the BEL character is sent to the TPD at the end of the operation. The TPD sounds the tone after the last character is displayed. When the flag is set to zero, the TPD does not sound the audible tone unless the warning beep flag is set.

6.5.3.4 Forced Termination Character. If this flag is set to one during a Read ASCII operation, a valid termination character must be received before the operation can complete. All characters received after the read buffer fills and before the termination character is received are discarded. If the flag is set to zero, the operation completes without receiving a termination character.

6.5.3.5 Echo. When this flag is set to zero, each character is printed as it is entered. When a key is pressed during Read ASCII operations and the flag is set to one, a blank is printed because the space character is substituted for the data character before it is returned to the TPD. When a key is pressed during Read Direct operations and the flag is set to one, no echo is produced. This flag applies only to Read ASCII and Read Direct operations. If the communications port was specified as no echo during system generation, this flag has no meaning, since echo is automatically suppressed.

6.5.3.6 Character Validation. This flag, when set to one, enables character validation of the field being read by a Read ASCII operation. Validation requires a validation table that specifies the characters to be accepted in the field. Character validation is discussed in greater detail in a subsequent paragraph. When the character validation flag is set to zero, no character validation is performed. Refer to the description of the Write ASCII operation for the use of the reply flag when the character validation flag is set to zero.

6.5.3.7 Validation Error Mode. The validation error mode flag, when set to one, enables correction of errors detected during validation of field contents by the task. The Validation Error Mode operation is effectively a Reread operation; the flags that apply to a Read apply in the same way to this operation. Only the error correction keys can be used. When a user reenters one or more characters in the field with an error correction key, the system sets the validation error mode flag to zero. When the calling task sets the validation error mode flag to zero, the operation is performed in the normal mode.

6.5.3.8 Warning Beep. When this flag is set to one, the BEL character is sent to the TPD and an audible tone sounds if an invalid function is requested.

6.5.3.9 Read ASCII Example. When a TPD has been opened in the event key mode and task edit keys are also enabled (task edit flag set to one), the task edit character is always returned in the event byte of the extended call block, and the event character is also returned in that byte in resource-specific I/O. The state of the event key flag in the system flag field indicates which type of character is in the event byte when both are enabled. The task accesses and decodes the character and performs the function corresponding to the key.

The following is an example of the code for a Read ASCII operation with event key termination enabled by the previous Open operation and task edit key termination enabled for the Read operation:

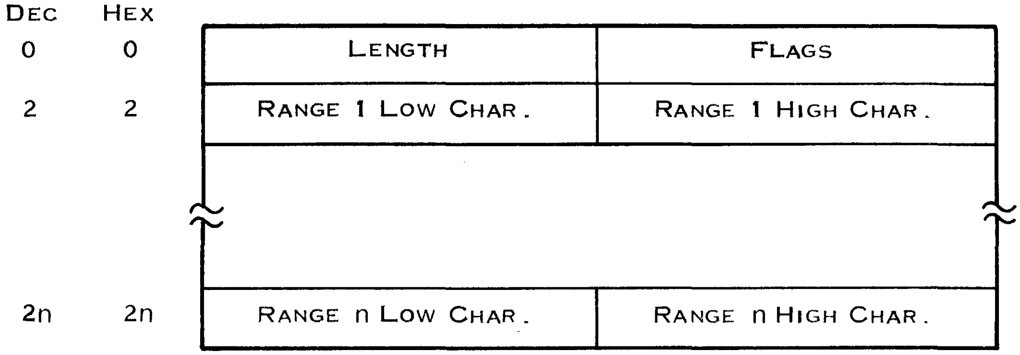
```
REHC     DATA 0              READ FIELD OF TERMINAL AT LUNO >3F.
         BYTE 9,>3F          FIELD SIZE IS 15 CHARACTERS.
SYSFL    BYTE 0              EVENT KEYS AND TASK EDIT KEYS
         BYTE >02            ENABLED.
         DATA RBUFF
         DATA 15
         DATA 0
         DATA 0
         DATA >0400
         DATA 0
         DATA 0
         DATA 0
```

6.5.3.10 Character Validation Operation. A Read ASCII operation can specify character validation by specifying a range of characters to be accepted or a range of characters to be rejected. If a range(s) of characters to be accepted is specified, the characters outside the range(s) read by the operation are rejected and characters within the range(s) specified are stored in the read buffer. If a range(s) of characters to be rejected is specified, characters within the range(s) read by the operation are rejected and characters outside the range(s) are stored in the read buffer. The character validation flag in the extended user flags field is set to one for a Read with Validation operation. The flags byte of the validation table specifies whether the specified ranges are for character acceptance or rejection.

Each Read operation with character validation must specify a validation table. Specifying a validation table requires:

- Setting the character validation flag to one
- Supplying a validation table
- Placing the address of the table in the character validation table address field of the extended call block

The validation table contains one or more ranges of characters that define the valid characters for the field. The table may define the valid characters by specifying ranges of characters that are not valid, or by specifying ranges of characters that are valid. Each range in the table requires two bytes, and the table contains two bytes of overhead. Thus the length of the table in bytes is two times the number of ranges, plus two. The format of the table is as follows:



DEC	HEX		
0	0	LENGTH	FLAGS
2	2	RANGE 1 LOW CHAR.	RANGE 1 HIGH CHAR.
2n	2n	RANGE n LOW CHAR.	RANGE n HIGH CHAR.

2283204

The validation table contains the following:

Byte	Contents
0	Length — Length of the validation table in bytes (2n + 2).
1	Flags: Bit 0 — Validation flag. Set as follows: 1 — Invalid ranges. Characters greater than or equal to the low character and less than or equal to the high character are invalid. 0 — Valid ranges. Characters greater than or equal to the low character and less than or equal to the high character are valid. Bits 1-7 — Reserved.
2	Low character for range 1.
3	High character for range 1.
	Character pairs for additional ranges.
2n	Low character for range n.
2n + 1	High character for range n.

Character validation is performed after each character is entered in the field.

When an invalid key is entered, the field enters character error mode. In character error mode, only the error correction keys operate, no echo occurs, and if Warning Beep is set, the BEL character is sent to the TPD after entry of a nonerror correction key.

The user must press one of the following correction keys when the last character entered is invalid:

- (CTRL) H or BACKSPACE
- RUB OUT
- (CTRL) N
- (CTRL) T

Next, the user enters the data correctly.

An example Read with Validation operation performs the following:

- Reads a ten-character field
- Validates the field as an alphanumeric field with no lowercase letters

The following is an example of the code for the supervisor call block and validation table for the example operation:

```
RVAL    DATA 0              READ FIELD OF TERMINAL AT LUNO >2B,
        BYTE 9,>2B          VALIDATING PER TABL. FIELD
        BYTE 0              SIZE IS 10 CHARACTERS. READ
        BYTE >02            BUFFER IS BUFF.
        DATA BUFF
        DATA 10
        DATA 0
        DATA TABL
FLG2    DATA >0004
        BSS 6
        .
        .
        .
        EVEN
TABL    BYTE 6              LENGTH OF TABLE
        BYTE 0              VALID RANGES
        DATA >3039          RANGE 1 — NUMERALS
        DATA >415A          RANGE 2 — UPPERCASE LETTERS
```

6.5.3.11 Field Validation. Any validation of a field must be performed by a task following the reading of the field. This could verify that the field contains the proper number of letters, followed by numbers, for example. The Read ASCII operation in the validation error mode is used by the task to obtain corrected data when an error has occurred. Character validation may be requested for the operation also. The validation error mode flag is set to one to enable the mode.

In the validation error mode, the operation requires reentry of the field. The correction key is the BACKSPACE key. When a character is entered prior to pressing the BACKSPACE key, the character is not printed (no echo).

The difference between a normal Read ASCII operation and one that specifies the validation error mode is that in the validation error mode input is ignored until the BACKSPACE key is pressed. The user should backspace to the leftmost error character and enter the correct characters in the remainder of the field.

The call block for the previous Read ASCII operation can be used by setting the validation error mode flag to one. The following instructions set the flag in the call block of the character validation coding example:

```
MASK2  BYTE  >2
       SOCB  @MASK2,@FLG2 + 1
```

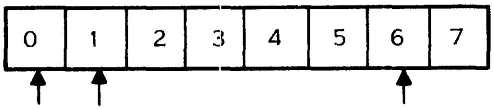
6.5.3.12 Getting Event Characters. The Remote Get Event Character operation (sub-opcode >05) returns an event character in the event byte (byte 17) of the extended supervisor call block. The LUNO assigned to the TPD does not have to be open. The operation is an alternative to performing a Read operation to obtain an event character.

The operation is a special type of Read Device Status operation; see previous explanation for this operation.

The following fields of the extended supervisor call block apply to a Remote Get Event Character operation:

- SVC code — 0
- Return code
- Sub-opcode — >05
- Logical unit number (LUNO)
- User flags
- Data buffer address
- Read character count
- <Actual read count>
- Event byte

The following user flags apply to a Remote Get Event Character operation:



2283205

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

Bit 1 — Reply flag. Set to one.

Bit 6 — Extended call flag. Set to one.

The logical unit number (LUNO) field contains the LUNO assigned to the device at which the event character is entered.

The data buffer address is the address of the buffer into which DNOS places the status information.

The read character count is the length of the buffer.

DNOS returns the number of characters stored in the buffer in the actual read count field.

DNOS returns the event character in the event byte.

The contents of the data buffer after a Remote Get Event Character operation has returned the status of a TPD is:

Byte	**Contents**
0–1	>FFFF.
2–3	Number of characters buffered in the input character queue.

DNOS maintains an input character queue that stores characters input while the system is processing a previously-entered character or command. Bytes 2 and 3 contain the number of characters in the queue. The maximum size of this queue is specified when the system is generated.

The following is an example of the source code for a supervisor call block for a Remote Get Event Character operation and code for the read buffer:

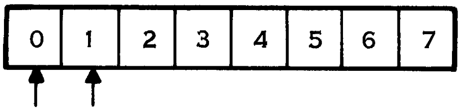
```
RGEVCH    DATA 0                        GET EVENT CHARACTER FROM TERMINAL
          BYTE 5,>32                    ASSIGNED TO LUNO >32.
          DATA >42
          DATA MRADR
          DATA 10
          DATA 0
          DATA 0
          DATA 0
          BYTE 0
EVCHAR    BYTE 0
          DATA 0,0
MRADR     BSS 1                         DEVICE STATUS BUFFER
          BSS 1
CHINQ     BSS 8
```

6.5.3.13 Read Direct. Sub-opcode >0A specifies a Read Direct operation. The Read Direct operation reads a record from the TPD and stores the characters in the specified buffer, two characters per word. The operation does not print (echo) the characters.

The following fields of the basic supervisor call block apply to a Read Direct operation:

- SVC code — 0
- Return code
- Sub-opcode — >0A
- Logical unit number (LUNO)
- <System flags>
- User flags
- Data buffer address
- Read character count
- <Actual read count>

The following system flags apply to a Read Direct operation:

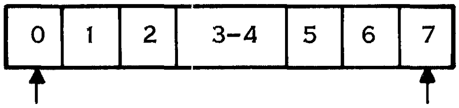


2279514

Bit 0 — Busy flag. Set by system as follows:
1 — Busy.
0 — Operation completed.

Bit 1 — Error flag. Set by system as follows:
1 — Error.
0 — No error.

The following user flags apply to a Read Direct operation:



2279515

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

Bit 7 — Blank adjustment flag. Set as follows:
1 — Read with blank adjustment.
0 — Read without blank adjustment.

The logical unit number (LUNO) field contains the LUNO assigned to the TPD from which a record is to be read.

The data buffer address is the address of the buffer into which DNOS places the record.

The read character count is the length of the buffer.

DNOS returns the number of characters stored in the buffer in the actual read count field.

The Read Direct operation recognizes the ASCII or JISCII codes as specified by the country code. The operation stores the characters in a word, packed one per byte. The most significant bit is set to zero for the seven-bit ASCII codes; all eight bits of the JISCII code are stored.

The Read Direct operation does not interpret characters. The operation terminates upon receipt of any of the following: current record terminator, EOF character, a specified number of characters, or expiration of the read time-out. The most significant bit of characters saved is set to zero (unless eight-bit data is specified).

If eight-bit ASCII is specified, all data passes directly to the task exactly as received from the device, without checking for parity or special characters. The operation terminates on receipt of the specified number of characters or expiration of the read time-out. If a termination character is specified (by setting the task edit flag and specifying the desired termination character in byte 17 of the call block), the operation also terminates upon receipt of a character matching the specified termination character. If the termination character is received, it is the last character in the read buffer.

If the device was previously in eight-bit ASCII mode, eight-bit ASCII is not specified, the device is restored to normal operating mode. If this method is used to reset the eight-bit ASCII mode, the next read operation may or may not check for parity, special characters, and so on.

When blank adjustment is specified for variable length records, blanks are stored in the buffer to fill the record. That is, when the record length is less than the buffer size, DNOS supplies blanks (>20) to fill the buffer. The character count returned in bytes 10 and 11 includes the blanks supplied by the system.

The following is an example of the source code for a supervisor call block for a Read Direct operation and code for the read buffer:

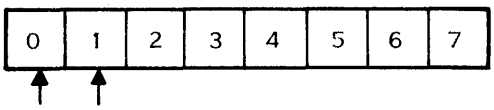
```
RDDHCT    DATA 0                   READ RECORD FROM TERMINAL ASSIGNED
          BYTE >A,>3E              TO LUNO >3E.
          BYTE 0,0
          DATA RDBUF
          DATA 80
          DATA 0
RDBUF     BSS 80                   READ BUFFER
```

6.5.3.14 Write Direct. Sub-opcode >0C specifies a Write Direct operation. The Write Direct operation transfers a record from the specified buffer to the TPD.

The following fields of the basic supervisor call block apply to a Write Direct operation:

- SVC code — 0
- Return code
- Sub-opcode — >0C
- Logical unit number (LUNO)
- <System flags>
- User flags
- Data buffer address
- Write character count

The following system flags apply to a Write Direct operation:

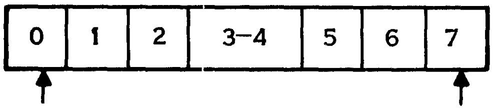


2283196

Bit 0 — Busy flag. Set by system as follows:
1 — Busy.
0 — Operation completed.

Bit 1 — Error flag. Set by system as follows:
1 — Error.
0 — No error.

The following user flags apply to a Write Direct operation:



2283192

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

Bit 7 — Blank adjustment flag. Set as follows:
1 — Write with blank adjustment.
0 — Write without blank adjustment.

The logical unit number (LUNO) field contains the LUNO assigned to the TPD to which a record is to be written.

The data buffer address is the address of the buffer that contains the record to be printed.

The write character count is the number of characters to be printed by the TPD.

The Write Direct operation prints a record on the TPD. The record consists of ASCII characters or JISCII characters, as specified by the country code.

Setting the eight-bit flag in the extended call block puts the device in the eight-bit ASCII mode. Setting the eight-bit ASCII flag to zero or using a standard call block puts the device in the normal operating mode. If the device had been transmitting data in the eight-bit ASCII mode previously, and this operation puts the device in the normal operating mode, the next read operation may or may not check and clear the parity bit on the characters received.

The ASCII characters from >00 through >1F are control and protocol characters and should be avoided in normal data traffic. The actions taken by a TPD depend on the specific terminal and how it is configured; refer to the manual for the terminal being used. When these characters are transmitted to the TPD these results occur:

- ENQ (>05) — The TPD transmits the answer back memory if it is defined.

- BEL (>07) — The bell rings.

- HT (>09) — The TPD prints a space.

- LF (>0A) — A line feed occurs.

- Form feed (>0C) — The TPD attempts a form feed if it is capable of performing a form feed.

- CR (>0D) — The TPD performs a carriage return.

- ESC (>1B) — Some TPD interpret this as the beginning of an Extended Device Control (EDC) sequence and attempt to interpret the following characters as meaningful terminal orders; the results can be unexpected.

When blank adjustment is specified, trailing blanks in the buffer are not written. The write character count in bytes 10 and 11 is not altered.

The following is an example of the source code for a supervisor call block for a Write Direct operation:

```
WAHCT   DATA 0              WRITE RECORD TO TERMINAL ASSIGNED TO
        BYTE >C,>4C         LUNO >4C INITIATE MODE.
        BYTE 0,>80
        DATA WRBUFF
        DATA 0
        DATA 80
```

6.6 PROGRAMMING FOR EVENT CHARACTERS

A set of event characters is defined for each type of terminal. These characters function as programmable function or edit keys. That is, the application program receives these characters, decodes them, and performs the desired function. The terminal must be opened with event keys enabled. Then, when an event key is pressed at the terminal, the event key flag in the system flags byte is set. For resource-specific I/O, the application program must execute a Remote Get Event Character operation to obtain the character. The Remote Get Event Character operation is described for each device.

6.7 CASSETTE I/O

DNOS supports resource-independent and resource-specific I/O for the cassette units of the 733 ASR. Resource-independent I/O for cassette units is analogous to I/O to magnetic tape units. Resource-specific I/O consists of the Read Direct and Write Direct operations.

The following I/O SVC block for cassette operations is the basic block used for all operations. If an extension to this block is necessary for a particular operation, it is indicated in the operation description.

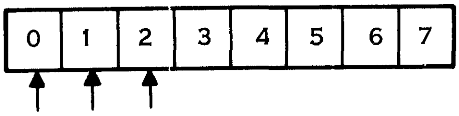
SVC >00 -- I/O Operations

Align on word boundary
Can be initiated as an
event

Dec	Hex		
0	0	>00	<Return Code>
2	2	Sub-Opcode	LUNO
4	4	<System Flags>	User Flags
6	6	Data Buffer Address	
8	8	Read Character Count	
10	A	Write Character Count/<Actual Read Count>	

2279470

The system flags (byte 4) in the supervisor call block apply to all cassette I/O. These flags are:



2279520

Bit 0 — Busy flag. Set by system as follows:
1 — Busy.
0 — Operation completed.

Bit 1 — Error flag. Set by system as follows:
1 — Error.
0 — No error.

Bit 2 — End-of-file. Set by system as follows:
1 — A DC3 (X-OFF) was read as the first character of a record.
0 — First character of record being read was not a DC3 (X-OFF).

Two user flags (byte 5) in the supervisor call block apply to all cassette I/O. However, significance of these flags differs for various operations. The flags that apply to each operation are described in the detailed description of each operation.

The operations appropriate for the cassette are described in subsequent paragraphs. The following sub-opcodes, which do not apply to the cassette, produce the indicated results:

05	Ignored
08	Error

6.7.1 Cassette Resource-Independent I/O

The subset of sub-opcodes for resource-independent I/O to cassettes is as follows:

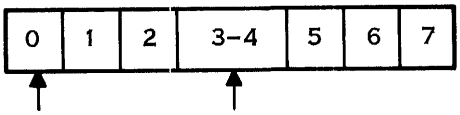
00	Open
01	Close
02	Close, Write EOF
03	Open and Rewind
04	Close and Unload
06	Forward Space
07	Backward Space
09	Read ASCII
0B	Write ASCII
0D	Write EOF
0E	Rewind
0F	Unload

6.7.1.1 Open. Sub-opcode >00 specifies an Open operation. The Open operation is required for the cassette units of a 733 ASR. However, whether or not DNOS validates the Open operation is specified when the system is generated. Validation does not allow an Open operation when the Open operation would result in a conflict with I/O to the same device by another task.

The following fields of the basic supervisor call block apply to an Open operation:

- SVC code — 0
- Return code
- Sub-opcode — >00
- Logical unit number (LUNO)
- User flags
- Data buffer address
- Read character count

The following flags in the user flag field apply to an Open operation:



2279521

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

Bits 3-4 — Access privilege flag. Set as follows:
00 — Exclusive write.
01 — Exclusive all.
10 — Shared.
11 — Read only.

The logical unit number (LUNO) field contains the LUNO assigned to the cassette to be opened.

The Open operation returns the device type code in the data buffer address field (bytes 6 and 7) of the supervisor call block. The device type code for the cassette is 3.

When the calling task places zero in the read character count field (bytes 8 and 9) of the supervisor call block, the Open operation returns the default logical record length for the device. The default logical record length for a cassette is >56.

The following is an example of the source code for a supervisor call block to open a cassette:

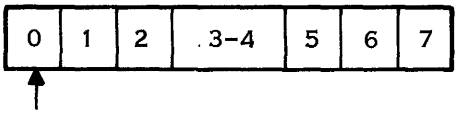
```
OCAS     DATA 0                  OPEN CASSETTE ASSIGNED TO LUNO >2A.
         BYTE 0,>2A
         DATA 0
TPE      DATA 0
RL       DATA 0
         DATA 0
```

6.7.1.2 Close. Sub-opcode >01 specifies a Close operation. The Close operation ends I/O to a LUNO from the calling task. The LUNO remains assigned to the device and may be opened again for additional I/O operations. When a task terminates, DNOS closes all LUNOs that have been opened by the task.

The following fields of the basic supervisor call block apply to a Close operation:

- SVC code — 0
- Return code
- Sub-opcode — >01
- Logical unit number (LUNO)
- User flags

The initiate flag in the user flag field applies to a Close operation:



2279522

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

The logical unit number (LUNO) field contains the LUNO to be closed.

The following is an example of the source code for a supervisor call block to close a cassette:

```
CCAS    DATA 0              CLOSE CASSETTE ASSIGNED TO LUNO >2A.
        BYTE 1,>2A
        DATA 0
        DATA 0
        DATA 0
        DATA 0
```

6.7.1.3 Close, Write EOF. The Close, Write EOF operation, sub-opcode >02, consists of a Write EOF operation followed by a Close operation.

6.7.1.4 Open and Rewind. The Open and Rewind operation, sub-opcode >03, is an Open operation followed by a Rewind operation.

6.7.1.5 Close and Unload. The Close and Unload operation, sub-opcode >04, consists of an Unload operation followed by a Close operation.

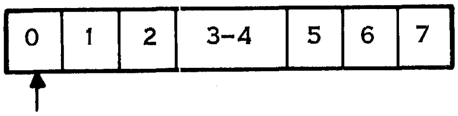
6.7.1.6 Forward Space. Sub-opcode >06 specifies a Forward Space operation. The Forward Space operation moves the tape forward a specified number of logical records or to the end-of-file record.

The following fields of the basic supervisor call block apply to a Forward Space operation:

- SVC code — 0
- Return code
- Sub-opcode — >06
- Logical unit number (LUNO)
- <System flags>
- User flags
- Write character count

The system flags defined for all cassette operations apply to a Forward Space operation.

The following user flag applies to a Forward Space operation:



2279523

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

The logical unit number (LUNO) field contains the LUNO assigned to the cassette to be forward spaced.

The write character count field (bytes 10 and 11) contains the number of logical records for the operation. The device service routine (DSR) stores a zero in the field when the tape is moved the specified number of records without reading an EOF record. When the operation reads an EOF record, the tape movement stops, and the number of records remaining to be moved is stored in the write character count field. The DSR also sets the EOF flag in the system flags byte.

The following is an example of the source code for a supervisor call block to forward space a cassette:

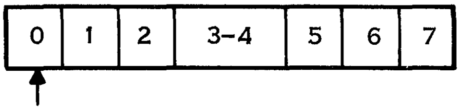
```
FSCS    DATA 0              FORWARD SPACE CASSETTE ASSIGNED TO
        BYTE >6,>4B         LUNO >4B THREE RECORDS
        DATA 0
        DATA 0
        DATA 0
        DATA 3
```

6.7.1.7 Backward Space. Sub-opcode >07 specifies a Backward Space operation. The Backward Space operation moves the tape in the reverse direction a specified number of logical records or to the end-of-file record.

The following fields of the basic supervisor call block apply to a Backward Space operation:

- SVC code — 0
- Return code
- Sub-opcode — >07
- Logical unit number (LUNO)
- <System flags>
- User flags
- Write character count

The following user flag applies to a Backward Space operation:



2279524

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

The logical unit number (LUNO) field contains the LUNO assigned to the cassette to be backward spaced.

The write character count field (bytes 10 and 11) contains the number of logical records for the operation. The device service routine (DSR) stores a zero in the field when the tape is moved the specified number of records without reading an EOF record. When the operation reads an EOF record, the tape movement stops, and the number of records remaining to be moved is stored in the write character count field. The DSR positions the tape so that the next Read operation reads the EOF record. The DSR also sets the EOF flag in the system flag byte.

The following is an example of the source code for a supervisor call block to backward space a cassette:

```
BSCS    DATA 0              BACKWARD SPACE CASSETTE ASSIGNED TO
        BYTE >7, >4B        LUNO >4B ONE RECORD
        DATA 0
        DATA 0
        DATA 0
        DATA 1
```

6.7.1.8 Read ASCII. Sub-opcode >09 specifies a Read ASCII operation. The Read ASCII operation reads a record from the cassette and stores the characters in the specified buffer, two characters per word.

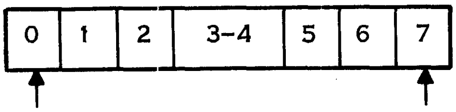
The following fields of the basic supervisor call block apply to a Read ASCII operation:

- SVC code — 0
- Return code
- Sub-opcode — >09
- Logical unit number (LUNO)
- <System flags>
- User flags
- Data buffer address

- Read character count
- <Actual read count>

The system flags defined for all cassette operations apply to a Read ASCII operation.

The following flags in the user flag field apply to a Read ASCII operation:



2279525

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

Bit 7 — Blank adjustment flag. Set as follows:
1 — Read with blank adjustment.
0 — Read without blank adjustment.

The logical unit number (LUNO) field contains the LUNO assigned to the cassette from which a record is to be read.

The data buffer address is the address of the buffer into which DNOS places the record.

The read character count is the length of the buffer.

DNOS returns the number of characters stored in the buffer in the input record length field.

The Read ASCII operation recognizes the characters listed in Appendix B for the cassette. The operation stores the characters, packed one per byte. When the country code in effect is not >0200 (Japan), the most significant bit of each character is set to zero. When the country code is >0200, the JISCII code applies. The device service routine (DSR) supplies the most significant bit for JISCII codes. A transition character transparent to the user is written between a code having zero as the most significant bit (seven-bit JISCII) and a code having one as the most significant bit. The maximum record length is a hardware requirement; the transition characters limit the length of the maximum record available to the user. The Read operation continues until a carriage return (>0D) character is read or the buffer is full. The maximum number of characters in a cassette record is 86.

When a carriage return is read, the number of characters read is stored in the actual read count field and the operation terminates.

When blank adjustment is specified for variable length records, blanks are stored in the buffer to fill the record. That is, when the record length is less than the buffer size, the device service routine (DSR) supplies blanks (>20) to fill the buffer. The character count returned in bytes 10 and 11 includes the blanks supplied by the DSR.

When a DC3 (X-OFF) character is read in the first character position, the device service routine (DSR) sets the EOF flag in the system flags byte, stores zero in the actual read count field, and terminates the operation.

The cassette unit does not provide a logical end-of-medium indication, but does provide a physical end-of-tape indication, which may occur at either end of the tape. The corresponding error code is returned by the operation that follows the detection of the physical end of tape.

The following is an example of the source code for a supervisor call block for a Read ASCII operation and the code for the read buffer:

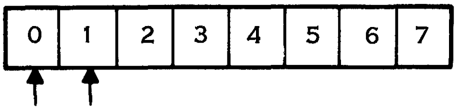
```
RDCAS     DATA 0                    READ RECORD FROM CASSETTE ASSIGNED
          BYTE 9,>2D                TO LUNO >2D IN THE INITIATE I/O
          BYTE 0,>80                MODE.
          DATA RB
          DATA 80
          DATA 0
RB        BSS 80                    READ BUFFER
```

6.7.1.9 Write ASCII. Sub-opcode >0B specifies a Write ASCII operation. The Write ASCII operation transfers a record from the specified buffer to the cassette.

The following fields of the basic supervisor call block apply to a Write ASCII operation:

- SVC code — 0
- Return code
- Sub-opcode — >0B
- Logical unit number (LUNO)
- <System flags>
- User flags
- Data buffer address
- Write character count

The following system flags apply to a Write ASCII operation:

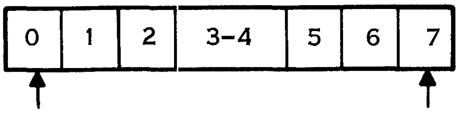


2279526

Bit 0 — Busy flag. Set by system as follows:
1 — Busy.
0 — Operation completed.

Bit 1 — Error flag. Set by system as follows:
1 — Error.
0 — No error.

The following user flags apply to a Write ASCII operation:



2279527

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

Bit 7 — Blank adjustment flag. Set as follows:
1 — Write with blank adjustment.
0 — Write without blank adjustment.

The logical unit number (LUNO) field contains the LUNO assigned to the cassette to which a record is to be written.

The data buffer address is the address of the buffer that contains the record to be written.

The write character count is the number of characters to be written on the cassette.

The Write ASCII operation writes a record on a cassette. The record consists of ASCII characters or JISCII characters as specified by the country code. Special considerations apply to the use of JISCII characters because the cassette units write a maximum of 86 seven-bit characters per record, including the characters written by the hardware. The device service routine (DSR) removes the most significant bit of the JISCII code, and the cassette writes the seven least significant bits. The DSR writes a transition character between JISCII code with zero as the most significant bit and JISCII code with one as the most significant bit. These transition characters (transparent to the user) allow the DSR to supply the most significant bit correctly when the record is read. However, they limit the number of characters the user can place in a record. For example, if no transition characters were required, the user task could write 80 characters per record; if two transitions occurred, 78 user-supplied characters would fill the record.

When the record being written contains a carriage return, the DSR replaces it with an end transmit block (ETB) character, >17. When the specified number of characters has been written, the DSR writes a carriage return (LCR), a line feed (LF), a record-off (DC4) character, >14, and a rubout (DEL) character, >7F.

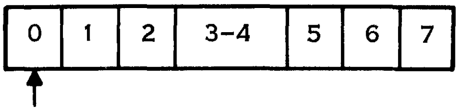
When blank adjustment is specified, trailing blanks in the buffer are not written. The output character count in bytes 10 and 11 is not altered.

6.7.1.10 Write EOF. Sub-opcode >0D specifies a Write EOF operation. The Write EOF operation writes an EOF record on the cassette tape.

The following fields of the basic supervisor call block apply to a Write EOF operation:

- SVC code — 0
- Return code
- Sub-opcode — >0D
- Logical unit number (LUNO)
- User flags

The following user flag applies to a Write EOF operation:



2279528

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

The logical unit number (LUNO) field contains the LUNO assigned to the cassette on which the EOF record is to be written.

The EOF record for a cassette consists of a DC3 character, a DC4 character, and a DEL character.

The following is an example of the source code for a supervisor call block to write an EOF record on a cassette:

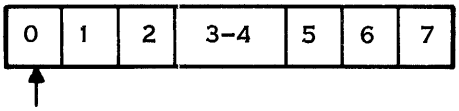
```
WECS    DATA 0              WRITE EOF ON CASSETTE ASSIGNED
        BYTE >D,>4B         TO LUNO >4B.
        DATA 0
        DATA 0
        DATA 0
        DATA 0
```

6.7.1.11 Rewind. Sub-opcode >0E specifies a Rewind operation. The Rewind operation rewinds the cassette to the clear area at the beginning of the tape, then moves the tape to the beginning-of-tape marker and lights the READY indicator on the 733 ASR.

The following fields of the basic supervisor call block apply to a Rewind operation:

- SVC code — 0
- Return code
- Sub-opcode — >0E
- Logical unit number (LUNO)
- User flags

The following user flag applies to a Rewind operation:



2279529

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

The logical unit number (LUNO) field contains the LUNO assigned to the cassette to be rewound.

The following is an example of the source code for a supervisor call block to rewind a cassette:

```
RWCS    DATA 0                  REWIND CASSETTE ASSIGNED TO LUNO >4B.
        BYTE >E,>4B
        DATA 0
        DATA 0
        DATA 0
        DATA 0
```

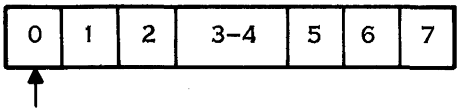
6.7.1.12 Unload. Sub-opcode >0F specifies an Unload operation. The Unload operation rewinds the cassette to the clear area at the beginning of the tape.

The following fields of the basic supervisor call block apply to an Unload operation:

- SVC code — 0
- Return code
- Sub-opcode — >0F

- Logical unit number (LUNO)
- User flags

The following user flag applies to an Unload operation:



2279530

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

The logical unit number (LUNO) field contains the LUNO assigned to the cassette to be unloaded.

The following is an example of the source code for a supervisor call block to unload a cassette:

```
ULCS     DATA 0              UNLOAD CASSETTE ASSIGNED TO
         BYTE >F,>4B         LUNO >4B
         DATA 0
         DATA 0
         DATA 0
         DATA 0
```

6.7.2 Cassette Resource-Specific I/O

The sub-opcodes for resource-specific I/O to cassettes are as follows:

0A	Read Direct
0C	Write Direct

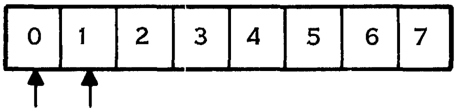
6.7.2.1 Read Direct. Sub-opcode >0A specifies a Read Direct operation. The Read Direct operation reads a record from the cassette and stores the characters in the specified buffer, two characters per word. The cassette unit transfers seven-bit characters to the computer; the DSR stores each character in the least significant bits of a byte, with a zero as the most significant bit.

The following fields of the basic supervisor call block apply to a Read Direct operation:

- SVC code — 0
- Return code
- Sub-opcode — >0A
- Logical unit number (LUNO)
- <System flags>
- User flags

- Data buffer address
- Read character count
- <Actual read count>

The following system flags apply to a Read Direct operation:

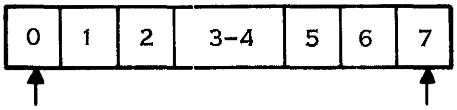


2279531

Bit 0 — Busy flag. Set by system as follows:
1 — Busy.
0 — Operation completed.

Bit 1 — Error flag. Set by system as follows:
1 — Error.
0 — No error.

The following flags in the user flag field apply to a Read Direct operation:



2279532

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

Bit 7 — Blank adjustment flag. Set as follows:
1 — Read with blank adjustment.
0 — Read without blank adjustment.

The logical unit number (LUNO) field contains the LUNO assigned to the cassette from which a record is to be read.

The data buffer address is the address of the buffer into which DNOS places the record.

The read character count is the length of the buffer.

DNOS returns the number of characters stored in the buffer in the actual read count field.

The Read Direct operation recognizes the ASCII or JISCII codes as specified by the country code. The operation stores the characters in a word, packed one per byte. The most significant bit of each character is set to zero for the seven-bit ASCII codes. The device service routine (DSR) supplies the most significant bit for JISCII codes. A transition character transparent to the user is written between a code having zero as the most significant bit (seven-bit JISCII) and a code having one as the most significant bit. The maximum record length is a hardware requirement; the transition characters limit the length of the maximum record available to the user. The Read Direct operation terminates when the buffer is full, a carriage return is read, or the maximum record (86 characters including control characters) is read.

When blank adjustment is specified for variable length records, blanks are stored in the buffer to fill the record. That is, when the record length is less than the buffer size, the device service routine (DSR) supplies blanks (>20) to fill the buffer. The character count returned in bytes 10 and 11 includes the blanks supplied by the DSR.

The following is an example of the source code for a supervisor call block for a Read Direct operation and code for the read buffer:

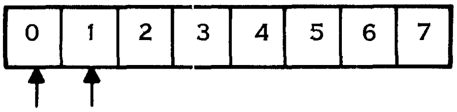
```
RDDCAS    DATA 0                      READ RECORD FROM CASSETTE ASSIGNED
          BYTE >A,>3C                 TO LUNO >3C.
          BYTE 0,0
          DATA RDBUF
          DATA 86
          DATA 0
          DATA 0
RDBUF     BSS 86                      READ BUFFER
```

6.7.2.2 Write Direct. Sub-opcode >0C specifies a Write Direct operation. The Write Direct operation transfers a record from the specified buffer to the cassette.

The following fields of the basic supervisor call block apply to a Write Direct operation:

- SVC code — 0
- Return code
- Sub-opcode — >0C
- Logical unit number (LUNO)
- <System flags>
- User flags
- Data buffer address
- Write character count

The following system flags apply to a Write Direct operation:

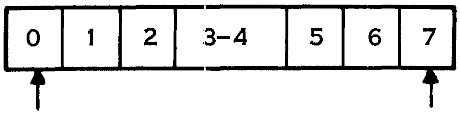


2279533

Bit 0 — Busy flag. Set by system as follows:
1 — Busy.
0 — Operation completed.

Bit 1 — Error flag. Set by system as follows:
1 — Error.
0 — No error.

The following user flags apply to a Write Direct operation:



2279534

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the

Bit 7 — Blank adjustment flag. Set as follows:
1 — Write with blank adjustment.
0 — Write without blank adjustment.

The logical unit number (LUNO) field contains the LUNO assigned to the cassette to which a record is to be written.

The data buffer address is the address of the buffer that contains the record to be written.

The write character count is the number of characters to be written on the cassette.

The Write Direct operation writes a record on a cassette. The record consists of ASCII characters or JISCII characters as specified by the country code. Special considerations apply to the use of JISCII characters because the cassette units write a maximum of 86 seven-bit characters per record, including the characters written by the hardware. The device service routine (DSR) removes the most significant bit of the JISCII code, and the cassette writes the seven least significant bits. The DSR writes a transition character between JISCII code with zero as the most significant bit and JISCII code with one as the most significant bit. These transition characters (transparent to the user) allow the DSR to supply the most significant bit correctly when the record is read. However, they limit the number of characters the user can place in a record. For example, if no transition characters were required, the user task could write 80 characters per record; if two transitions occurred, 78 user-supplied characters would fill the record.

When the specified number of characters has been written, the DSR writes a carriage return, a line feed, a record-off (DC4) character, >14, and a rubout (DEL) character, >7F. When the record contains a DC4 character, the DSR writes the DC4 character followed by a record-on (DC2) character.

The last record must terminate with a carriage return to assure that the record is actually written on the cassette.

When blank adjustment is specified, trailing blanks in the buffer are not written. The output character count in bytes 10 and 11 is not altered.

The following is an example of the source code for a supervisor call block for a Write Direct operation:

```
WACAS   DATA 0              WRITE RECORD TO CASSETTE ASSIGNED TO
        BYTE >C,>4C         LUNO >4C INITIATE MODE.
        BYTE 0,>80
        DATA WRBUFF
        DATA 0
        DATA 80
```

6.8 PRINTER OUTPUT

DNOS supports resource-independent I/O for the printer. Resource-independent I/O for the printer includes most I/O operations except the Read operations.

The subset of sub-opcodes for the printer applies, as follows:

00	Open
01	Close
02	Close, Write EOF
03	Open and Rewind
04	Close and Unload
05	Read Device Characteristics
0B	Write ASCII
0C	Write Direct
0D	Write EOF
0E	Rewind

The following I/O SVC block for printer operations is the basic block used for all operations. If an extension to this block is necessary for a particular operation, it is indicated in the operation description.

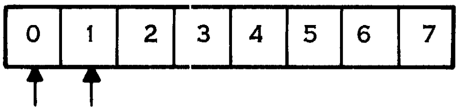
SVC >00 -- I/O OPERATIONS

ALIGN ON WORD BOUNDARY
CAN BE INITIATED AS AN EVENT

DEC	HEX		
0	0	>00	<RETURN CODE>
2	2	SUB-OPCODE	LUNO
4	4	<SYSTEM FLAGS>	USER FLAGS
6	6	DATA BUFFER ADDRESS	
8	8	READ CHARACTER COUNT	
10	A	WRITE CHARACTER COUNT/<ACTUAL READ COUNT>	

2279470

The system flags (byte 4) in the supervisor call block apply to all printer I/O. These flags are:



2279535

Bit 0 — Busy flag. Set by system as follows:
1 — Busy.
0 — Operation completed.

Bit 1 — Error flag. Set by system as follows:
1 — Error.
0 — No error.

One user flag (byte 5) in the supervisor call block applies to printer I/O. However, significance of these flags differs for various operations. The flags that apply to each operation are described in the detailed description of each operation.

The operations appropriate for the printer are described in subsequent paragraphs. The following sub-opcodes, which do not apply to the printer, produce the indicated results:

06	Ignored
07	Ignored
08	Error
09	Error
0A	Error
0F	Ignored

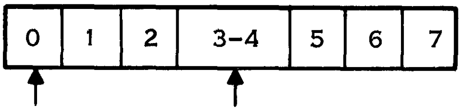
6.8.1 Open

Sub-opcode >00 specifies an Open operation. The Open operation is required for the printer. However, whether or not DNOS validates the Open operation is specified when the system is generated. Validation does not allow an Open operation when the Open operation would result in a conflict with I/O to the same device by another task.

The following fields of the basic supervisor call block apply to an Open operation:

- SVC code — 0
- Return code
- Sub-opcode — >00
- Logical unit number (LUNO)
- User flags
- Data buffer address
- Read character count

The following flags in the user flag field apply to an Open operation:



2279536

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

Bits 3-4 — Access privilege flag. Set as follows:
00 — Exclusive write.
01 — Exclusive all.
10 — Shared.
11 — Read only.

The logical unit number (LUNO) field contains the LUNO assigned to the printer to be opened.

The Open operation returns the device type code in the data buffer address field (bytes 6 and 7) of the supervisor call block. The device type code for the printer is 2.

When the calling task places zero in the read character count field (bytes 8 and 9) of the supervisor call block, the Open operation returns the default logical record length for the device. The default logical record length for a printer is >86.

An Open operation causes the printer to perform a carriage return and a line feed operation.

The following is an example of the source code for a supervisor call block to open a printer:

```
OLP     DATA 0             OPEN LINE PRINTER ASSIGNED
        BYTE 0,>2C         TO LUNO >2C
        DATA 0
TYP     DATA 0
LEN     DATA 0
        DATA 0
```

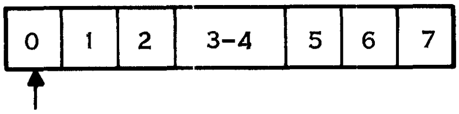
6.8.2 Close

Sub-opcode >01 specifies a Close operation. The Close operation ends I/O to a LUNO from the calling task. The LUNO remains assigned to the device, and may be opened again for additional I/O operations. When a task terminates, DNOS closes all LUNOs that have been opened by the task. A Close operation causes the printer to perform a carriage return.

The following fields of the basic supervisor call block apply to a Close operation:

- SVC code — 0
- Return code
- Sub-opcode — >01
- Logical unit number (LUNO)
- User flags

The initiate flag in the user flag field applies to a Close operation:



2279537

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

The logical unit number (LUNO) field contains the LUNO to be closed.

The following is an example of the source code for a supervisor call block to close a printer:

```
CLP     DATA 0             CLOSE LINE PRINTER ASSIGNED
        BYTE 1,>2C         TO LUNO >2C
        DATA 0
        DATA 0
        DATA 0
        DATA 0
```

6.8.3 Close, Write EOF

The Close, Write EOF operation, sub-opcode >02, consists of a Write EOF operation followed by a Close operation. The Write EOF operation for the printer is a form feed operation.

6.8.4 Open and Rewind

The Open and Rewind operation, sub-opcode >03, is an Open operation followed by a Rewind operation.

6.8.5 Close and Unload

On a printer, the Close and Unload operation, sub-opcode >04, is identical to a Close EOF operation.

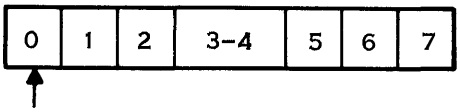
6.8.6 Read Device Characteristics

Sub-opcode >05 specifies a Read Device Characteristics operation. The Read Device Characteristics operation returns into the data buffer as much of the device information block as specified (up to a maximum of 38 characters) by the user in the read character count field.

The following fields of the basic supervisor call block apply to a Read Device Characteristics operation:

- SVC code — 0
- Return code
- Sub-opcode — >05
- Logical unit number (LUNO)
- User flags
- Data buffer address
- Read character count
- <Actual read count>

The following user flag applies to a Read Device Characteristics operation:



2279537

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

The logical unit number (LUNO) field contains the LUNO assigned to the device for which status information is returned.

The data buffer address is the address of the buffer into which DNOS places the status information.

The read character count is the number of characters of device information desired.

DNOS returns the number of characters stored in the buffer in the actual read count field. The number of characters cannot exceed 38.

The contents of the data buffer after a Read Device Characteristics operation has returned the status of a printer are in Table 6-5.

Table 6-5. Characteristics of Printer Devices

Byte	Bit	Contents
0		Reserved (>00)
1		Reserved (>FF)
2,3		Reserved (>0000)
4		DSR type (>01 indicates any printer except one that uses a CI403 or CI404 board; >31 indicates an attached printer)
5		Reserved (>00)
6,7		CRU/TILINE address of hardware interface
8,9		Reserved (>FFFF)
10		Hardware interface type >05 = TTY/EIA >06 = a 9902 port >80 = parallel printer
11-21		Reserved (>00)
22		Line flags
	0	Reserved (0)
	1	Switched line (defined during system generation)
	2-7	Reserved (0)
23		Reserved (>00)
24		Speed code 0 = 50 baud 1 = 75 baud 2 = 110 baud 3 = 134.5 baud 4 = 150 baud 5 = 200 baud 6 = 300 baud 7 = 600 baud 8 = 1200 baud 9 = 1800 baud A = 2400 baud B = 3600 baud C = 4800 baud D = 7200 baud E = 9600 baud F = 14400 baud 10 = 19200 baud 11 = 28800 baud 12 = 38400 baud

Table 6-5. Characteristics of Printer Devices (Continued)

Byte	Bit	Contents
25-37		Reserved (>00)

The buffer has the following format for printers that use a CI403 or CI404 board:

Byte	Bit	Contents
0		Reserved (>00)
1		Reserved (>FF)
2,3		Reserved (>0000)
4		DSR type (>10 indicates a printer that uses a CI403 or CI404 board)
5		Port identification number
6,7		CRU/TILINE address of hardware interface
8,9		Reserved (>FFFF)
10		Hardware interface type
		>23 = CI403 port
		>24 = CI404 port
11-21		Reserved (>00)
22		Line flags
	0	Half duplex
	1	Switched line (defined during system generation)
	2-7	Reserved (0)
23		Reserved (>00)
24		Speed code
		0 = 50 baud
		1 = 75 baud
		2 = 110 baud
		3 = 134.5 baud
		4 = 150 baud
		5 = 200 baud
		6 = 300 baud
		7 = 600 baud
		8 = 1200 baud
		9 = 1800 baud
		A = 2400 baud
		B = 3600 baud
		C = 4800 baud
		D = 7200 baud
		E = 9600 baud
		F = 14400 baud
		10 = 19200 baud
		11 = 28800 baud
		12 = 38400 baud
25-37		Reserved (>00)

The following is an example of the source code for a supervisor call block for a Read Device Characteristics operation and the read buffer. This call block returns the first 10 characters of the device information block (through the ACU CRU address). The number of characters currently buffered in the input queue is returned as part of this status information at the label LIQ.

```
RDSHCT   DATA 0                       READ STATUS OF PRINTER ASSIGNED TO
         BYTE 5,>35                   LUNO >35.
         DATA 0
         DATA DMY
         DATA 10
         DATA 0
DMY      BSS 2                        DEVICE STATUS BUFFER
LIQ      BSS 8
```

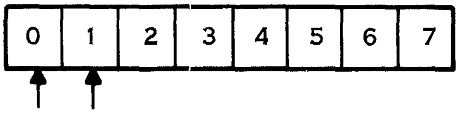
6.8.7 Write ASCII

Sub-opcode >0B specifies a Write ASCII operation. The Write ASCII operation transfers a record from the specified buffer to the printer.

The following fields of the basic supervisor call block apply to a Write ASCII operation:

- SVC code — 0
- Return code
- Sub-opcode — >0B
- Logical unit number (LUNO)
- <System flags>
- User flags
- Data buffer address
- Write character count

The following system flags apply to a Write ASCII operation:

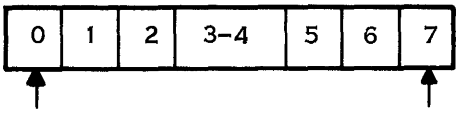


2279538

Bit 0 — Busy flag. Set by system as follows:
1 — Busy.
0 — Operation completed.

Bit 1 — Error flag. Set by system as follows:
1 — Error.
0 — No error.

The following user flags apply to a Write ASCII operation:



2279539

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

Bit 7 — Blank adjustment flag. Set as follows:
1 — Print with blank adjustment.
0 — Print without blank adjustment.

The logical unit number (LUNO) field contains the LUNO assigned to the printer.

The data buffer address is the address of the buffer that contains the record to be printed.

The write character count is the number of characters to be printed.

The Write ASCII operation prints a line on the printer. The line consists of ASCII characters or JIS-CII characters as specified by the country code.

All printers support special programmable features that increase the capability of each printer. For further information about these features refer to the appropriate printer manual.

When blank adjustment is specified, trailing blanks in the buffer are not printed. The write character count in bytes 10 and 11 is not altered.

The following is an example of the source code for a supervisor call block for a Write ASCII operation:

```
WALP    DATA 0              PRINT LINE ON LINE PRINTER ASSIGNED TO
        BYTE >B,>3C         LUNO >3C INITIATE MODE.
        BYTE 0,>80
        DATA WRBUFF
        DATA 0
        DATA 80
```

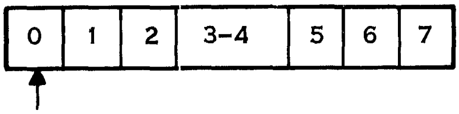
6.8.8 Write Direct

Sub-opcode >0C specifies a Write Direct operation. The Write Direct operation transfers a record from the specified buffer to the printer.

The following fields of the basic supervisor call block apply to a Write Direct operation:

- SVC code — 0
- Return code
- Sub-opcode — >0B
- Logical unit number (LUNO)
- <System flags>
- User flags
- Data buffer address
- Write character count

The following system flags apply to a Write Direct operation:

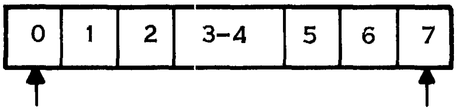


2279540

Bit 0 — Busy flag. Set by system as follows:
1 — Busy.
0 — Operation completed.

Bit 1 — Error flag. Set by system as follows:
1 — Error.
0 — No error.

The following user flags apply to a Write Direct operation:



2279539

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

Bit 7 — Blank adjustment flag. Set as follows:
1 — Print with blank adjustment.
0 — Print without blank adjustment.

The logical unit number (LUNO) field contains the LUNO assigned to the printer.

The data buffer address is the address of the buffer that contains the record to be printed.

The write character count is the number of characters to be printed.

The Write Direct operation prints a line on the printer. The line consists of ASCII characters or JISCII characters as specified by the country code.

All printers support special programmable features that increase the capability of each printer. For further information about these features refer to the appropriate printer manual.

When blank adjustment is specified, trailing blanks in the buffer are not printed. The write character count in bytes 10 and 11 is not altered.

The following is an example of the source code for a supervisor call block for a Write Direct operation:

```
WALP     DATA 0               PRINT LINE ON LINE PRINTER ASSIGNED TO
         BYTE >B,>3C          LUNO >3C INITIATE MODE.
         BYTE 0,>80
         DATA WRBUFF
         DATA 0
         DATA 80
```

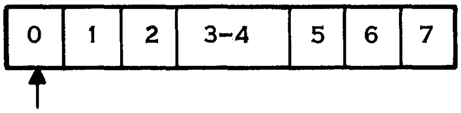
6.8.9 Write EOF

Sub-opcode >0D specifies a Write EOF operation. The Write EOF operation performs a form feed operation on the printer.

The following fields of the basic supervisor call block apply to a Write EOF operation:

- SVC code — 0
- Return code
- Sub-opcode — >0D
- Logical unit number (LUNO)
- User flags

The following user flag applies to a Write EOF operation:



2279540

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

The logical unit number (LUNO) field contains the LUNO assigned to the printer.

The following is an example of the source code for a supervisor call block to write an EOF to a printer:

```
WELP     DATA 0                       WRITE EOF TO LINE PRINTER ASSIGNED
         BYTE >D,>2C                  TO LUNO >2C
         DATA 0
         DATA 0
         DATA 0
         DATA 0
```

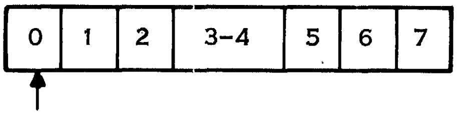
6.8.10 Rewind

Sub-opcode >0E specifies a Rewind operation. The Rewind operation performs a form feed operation on a printer.

The following fields of the basic supervisor call block apply to a Rewind operation:

- SVC code — 0
- Return code
- Sub-opcode — >0E
- Logical unit number (LUNO)
- User flags

The following user flag applies to a Rewind operation:



2279541

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

The logical unit number (LUNO) field contains the LUNO assigned to the printer.

The following is an example of the source code for a supervisor call block to rewind a printer:

```
RWLP     DATA 0                       REWIND LINE PRINTER ASSIGNED
         BYTE >E,>3E                  TO LUNO >3E
         DATA 0
         DATA 0
         DATA 0
         DATA 0
```

6.9 MAGNETIC TAPE I/O

DNOS supports both resource-independent and resource-specific I/O for the Model 979A Magnetic Tape Units. Except for the Read Device Status operation, the device must be opened using sub-opcode >00 or >03 prior to any I/O operation.

The following I/O SVC block for magnetic tape operations is the basic block used for all operations. If an extension to this block is necessary for a particular operation, it is indicated in the operation description.

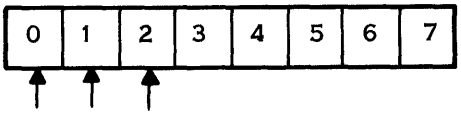
SVC >00 -- I/O OPERATIONS

ALIGN ON WORD BOUNDARY
CAN BE INITIATED AS AN EVENT

DEC	HEX		
0	0	>00	<RETURN CODE>
2	2	SUB-OPCODE	LUNO
4	4	<SYSTEM FLAGS>	USER FLAGS
6	6	DATA BUFFER ADDRESS	
8	8	READ CHARACTER COUNT	
10	A	WRITE CHARACTER COUNT/<ACTUAL READ COUNT>	

2279470

The system flags (byte 4) in the supervisor call block apply to all magnetic tape I/O. These flags are:



2279542

Bit 0 — Busy flag. Set by system as follows:
1 — Busy.
0 — Operation completed.

Bit 1 — Error flag. Set by system as follows:
1 — Error.
0 — No error.

Bit 2 — End-of-file. Set by system as follows:
1 — An EOF mark was read on the tape.
0 — A read operation did not read an EOF mark.

The user flags (byte 5) in the supervisor call block apply to all magnetic tape I/O. However, significance of these flags differs for various operations. The flags that apply to each operation are described in the detailed description of each operation.

The operations appropriate for magnetic tape are described in subsequent paragraphs. The following sub-opcode does not apply to magnetic tape; it produces the indicated result:

08	Error

6.9.1 Magnetic Tape Resource-Independent I/O

The sub-opcodes for resource-independent I/O to magnetic tape units are as follows:

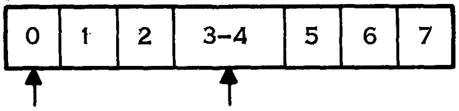
00	Open
01	Close
02	Close, Write EOF
03	Open and Rewind
04	Close and Unload
05	Read Device Status
06	Forward Space
07	Backward Space
09	Read ASCII
0B	Write ASCII
0D	Write EOF
0E	Rewind
0F	Unload

6.9.1.1 Open. Sub-opcode >00 specifies an Open operation. The Open operation is required for a magnetic tape transport. However, DNOS does not validate the Open operation; that is, it does not detect a possible conflict with I/O to the same device by another task. An Open operation is not required prior to performing a Read Device Status operation.

The following fields of the basic supervisor call block apply to an Open operation:

- SVC code — 0
- Return code
- Sub-opcode — >00
- Logical unit number (LUNO)
- User flags
- Data buffer address
- Read character count

The following flags in the user flag field apply to an Open operation:



2279543

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

Bits 3-4 — Access privilege flag. Set as follows:
00 — Exclusive write.
01 — Exclusive all.
10 — Shared.
11 — Read only.

The logical unit number (LUNO) field contains the LUNO assigned to the magnetic tape transport to be opened.

The Open operation returns the device type code in the data buffer address field (bytes 6 and 7) of the supervisor call block. The device type code for the magnetic tape is 8.

The following is an example of the source code for a supervisor call block to open a magnetic tape transport:

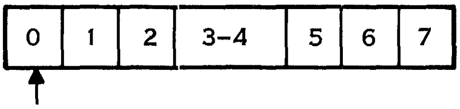
```
OMT     DATA 0                    OPEN MAG TAPE ASSIGNED TO LUNO >2F.
        BYTE 0,>2F
        DATA 0
MTT     DATA 0
        DATA 0
        DATA 0
```

6.9.1.2 Close. Sub-opcode >01 specifies a Close operation. The Close operation ends I/O to a LUNO from the calling task. The LUNO remains assigned to the device and may be opened again for additional I/O operations. When a task terminates, DNOS closes all LUNOs that have been opened by the task.

The following fields of the basic supervisor call block apply to a Close operation:

- SVC code — 0
- Return code
- Sub-opcode — >01
- Logical unit number (LUNO)
- User flags

The initiate flag in the user flag field applies to a Close operation:



2279544

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

The logical unit number (LUNO) field contains the LUNO to be closed.

The following is an example of the source code for a supervisor call block to close a magnetic tape:

```
CMT     DATA 0                  CLOSE MAG TAPE ASSIGNED TO LUNO >2F.
        BYTE 1,>2F
        DATA 0
        DATA 0
        DATA 0
        DATA 0
```

6.9.1.3 Close, Write EOF. The Close, Write EOF operation, sub-opcode >02, consists of a Write EOF operation followed by a Close operation.

6.9.1.4 Open and Rewind. The Open and Rewind operation, sub-opcode >03, is an Open operation followed by a Rewind operation.

6.9.1.5 Close and Unload. The Close and Unload operation, sub-opcode >04, consists of an Unload operation followed by a Close operation.

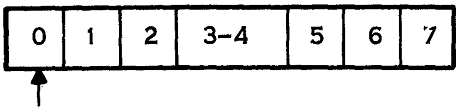
6.9.1.6 Read Device Status. Sub-opcode >05 specifies a Read Device Status operation. The Read Device status operation returns two bytes of status information.

The following fields of the basic supervisor call block apply to a Read Device Status operation:

- SVC code — 0
- Return code
- Sub-opcode — >05
- Logical unit number (LUNO)
- User flags
- Data buffer address

- Read character count
- <Actual read count>

The following user flag applies to a Read Device Status operation:



2279545

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

The logical unit number (LUNO) field contains the LUNO assigned to the magnetic tape transport.

The data buffer address is the address of the buffer into which DNOS places the status information.

The read character count is the length of the buffer.

DNOS returns the number of characters stored in the buffer in the actual read count field. The system returns >0002 in this field when the specified LUNO is assigned to a magnetic tape transport.

After a Read Device Status operation for a magnetic tape transport, the data buffer contains two bytes of information:

Byte	Contents
0	Transport status: >80 — Online, write ring installed on tape reel. >40 — Online, no write ring. >20 — Offline.
1	Recording mode: >80 — Phase encoded, 1600 or 3200 bits per inch (bpi). 00 — Nonreturn-to-zero inverted (NRZI), 800 bpi.

The following is an example of the source code for a supervisor call block for a Read Device Status operation and code for the read buffer:

```
RDSMT   DATA 0              READ STATUS OF MAG TAPE ASSIGNED TO
        BYTE 5,>3C          LUNO >3C.
        DATA 0
        DATA ST
        DATA 10
STL     DATA 0
ST      BSS 10              DEVICE STATUS BUFFER
```

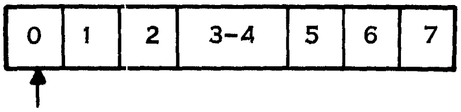
6.9.1.7 Forward Space. Sub-opcode >06 specifies a Forward Space operation. The Forward Space operation moves the tape forward a specified number of logical records or to the end-of-file record.

The following fields of the basic supervisor call block apply to a Forward Space operation:

- SVC code — 0
- Return code
- Sub-opcode — >06
- Logical unit number (LUNO)
- <System flags>
- User flags
- Write character count

The system flags defined for all magnetic tape operations apply to a Forward Space operation.

The following user flag applies to a Forward Space operation:



2279546

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

The logical unit number (LUNO) field contains the LUNO assigned to the magnetic tape to be forward spaced.

The write character count field (bytes 10 and 11) contains the number of logical records for the operation. The device service routine (DSR) stores a zero in the field when the tape is moved the specified number of records without reading an EOF record. When the operation reads an EOF record, the tape movement stops, and the number of records remaining to be moved is stored in the write character count field. The DSR also sets the EOF flag in the system flags byte. When the end of the tape is reached, the operation receives notification in the SVC call block.

The following is an example of the source code for a supervisor call block to forward space a magnetic tape:

```
FSMT      DATA 0                    FORWARD SPACE MAG TAPE ASSIGNED TO
          BYTE >6,>3B               LUNO >3B FIVE RECORDS
          DATA 0
          DATA 0
          DATA 0
          DATA 5
```

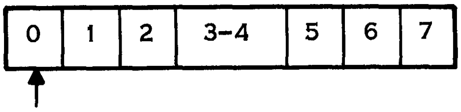
6.9.1.8 Backward Space. Sub-opcode >07 specifies a Backward Space operation. The Backward Space operation moves the tape in the reverse direction a specified number of logical records or to the end-of-file record.

The following fields of the basic supervisor call block apply to a Backward Space operation:

- SVC code — 0
- Return code
- Sub-opcode — >07
- Logical unit number (LUNO)
- <System flags>
- User flags
- Write character count

The system flags defined for all magnetic tape operations apply to a Backward Space operation.

The following user flag applies to a Backward Space operation:



2279547

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

The logical unit number (LUNO) field contains the LUNO assigned to the magnetic tape to be backward spaced.

The write character count field (bytes 10 and 11) contains the number of logical records for the operation. The device service routine (DSR) stores a zero in the field when the tape is moved the specified number of records without reading an EOF record. When the operation reads an EOF record, the tape movement stops, and the number of records remaining to be moved is stored in the write character count field. The DSR positions the tape so that the next Read operation reads the EOF record. The DSR also sets the EOF flag in the system flag byte. When the beginning of the tape is reached, the operation receives notification in the SVC call block.

The following is an example of the source code for a supervisor call block to backward space a magnetic tape:

```
BSMT    DATA 0              BACKWARD SPACE MAG TAPE ASSIGNED TO
        BYTE >7,>3B         LUNO >3B FOUR RECORDS
        DATA 0
        DATA 0
        DATA 0
        DATA 4
```

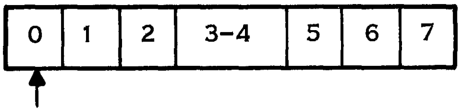
6.9.1.9 Read ASCII. Sub-opcode >09 specifies a Read ASCII operation. The Read ASCII operation reads a record from the magnetic tape and stores the characters in the specified buffer, one character per byte. If a Read ASCII operation is attempted at the end-of-tape, data is stored in the buffer and an error code is returned.

The following fields of the basic supervisor call block apply to a Read ASCII operation:

- SVC code — 0
- Return code
- Sub-opcode — >09
- Logical unit number (LUNO)
- <System flags>
- User flags
- Data buffer address
- Read character count
- <Actual read count>

The system flags defined for all magnetic tape operations apply to a Read ASCII operation.

The initiate flag in the user flag field applies to a Read ASCII operation:



2279548

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

The logical unit number (LUNO) field contains the LUNO assigned to the magnetic tape from which a record is to be read.

The data buffer address is the address of the buffer into which DNOS places the record.

The read character count is the length of the buffer.

DNOS returns the number of characters stored in the buffer in the actual read count field.

The Read ASCII operation reads the characters listed in Appendix B for magnetic tape. The operation stores the characters, packed one per byte. The Read operation continues to the end of the record, or until the buffer is full.

When the record has been read, the number of characters read is stored in the actual read count field and the operation terminates.

When an EOF mark is read, the device service routine (DSR) sets the EOF flag in the system flags byte, stores zero in the actual read count field, and terminates the operation.

The following is an example of the source code for a supervisor call block for a Read ASCII operation and code for the read buffer:

```
RDMT     DATA 0              READ RECORD FROM MAG TAPE ASSIGNED
         BYTE 9,>3D          TO LUNO >3D IN THE INITIATE I/O
         BYTE 0,>80          MODE.
         DATA RDB
         DATA 150
         DATA 0
RDB      BSS 150             READ BUFFER
```

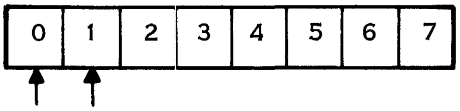
6.9.1.10 Write ASCII. Sub-opcode >0B specifies a Write ASCII operation. The Write ASCII operation transfers a record from the specified buffer to the magnetic tape. If a Write ASCII operation is attempted at the end-of-tape, the specified buffer is written with an error code returned.

The following fields of the basic supervisor call block apply to a Write ASCII operation:

- SVC code — 0
- Return code

- Sub-opcode — >0B
- Logical unit number (LUNO)
- <System flags>
- User flags
- Data buffer address
- Write character count

The following system flags apply to a Write ASCII operation:

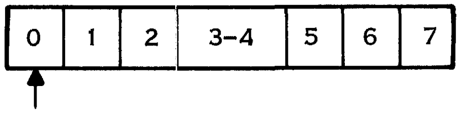


2279549

Bit 0 — Busy flag. Set by system as follows:
1 — Busy.
0 — Operation completed.

Bit 1 — Error flag. Set by system as follows:
1 — Error.
0 — No error.

The following user flags apply to a Write ASCII operation:



2279550

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

The logical unit number (LUNO) field contains the LUNO assigned to the magnetic tape to which a record is to be written.

The data buffer address is the address of the buffer that contains the record to be written.

The write character count is the number of characters to be written on the magnetic tape.

The Write ASCII operation writes a record on the magnetic tape. The characters in the buffer are not translated.

The following is an example of the source code for a supervisor call block for a Write ASCII operation:

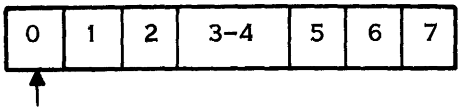
```
WAMT     DATA 0               WRITE RECORD TO MAG TAPE ASSIGNED TO
         BYTE >B,>3C          LUNO >3C INITIATE MODE.
         BYTE 0,>80
         DATA WRBUFF
         DATA 0
         DATA 60
```

6.9.1.11 Write EOF. Sub-opcode >0D specifies a Write EOF operation. The Write EOF operation writes an EOF mark on the magnetic tape.

The following fields of the basic supervisor call block apply to a Write EOF operation:

- SVC code — 0
- Return code
- Sub-opcode — >0D
- Logical unit number (LUNO)
- User flags

The following user flag applies to a Write EOF operation:



2279551

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

The logical unit number (LUNO) field contains the LUNO assigned to the magnetic tape on which the EOF mark is to be written.

The following is an example of the source code for a supervisor call block to write an EOF mark on a magnetic tape:

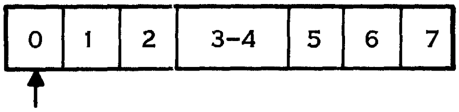
```
WEMT     DATA 0               WRITE EOF ON MAG TAPE ASSIGNED
         BYTE >D,>3B          TO LUNO >3B.
         DATA 0
         DATA 0
         DATA 0
         DATA 0
```

6.9.1.12 Rewind. Sub-opcode >0E specifies a Rewind operation. The Rewind operation rewinds the magnetic tape to the load point and places the transport in the ready state.

The following fields of the basic supervisor call block apply to a Rewind operation:

- SVC code — 0
- Return code
- Sub-opcode — >0E
- Logical unit number (LUNO)
- User flags

The following user flag applies to a Rewind operation:



2279552

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

The logical unit number (LUNO) field contains the LUNO assigned to the magnetic tape to be rewound.

The following is an example of the source code for a supervisor call block to rewind a magnetic tape:

```
RWMT     DATA 0                   REWIND MAG TAPE ASSIGNED TO LUNO >4D.
         BYTE >E,>4D
         DATA 0
         DATA 0
         DATA 0
         DATA 0
```

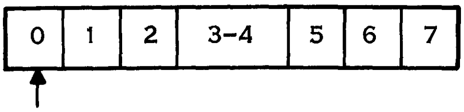
6.9.1.13 Unload. Sub-opcode >0F specifies an Unload operation. The Unload operation rewinds the magnetic tape to the physical beginning of the tape, leaving the tape reel ready for removal. No more I/O to the drive is allowed until the tape reel is remounted or another tape reel is mounted on the transport.

The following fields of the basic supervisor call block apply to an Unload operation:

- SVC code — 0
- Return code

- Sub-opcode — >0F
- Logical unit number (LUNO)
- User flags

The following user flag applies to an Unload operation:



2279553

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

The logical unit number (LUNO) field contains the LUNO assigned to the magnetic tape to be unloaded.

The following is an example of the source code for a supervisor call block to unload a magnetic tape:

```
ULMT    DATA 0                    UNLOAD MAG TAPE ASSIGNED TO LUNO >42.
        BYTE >F,>42
        DATA 0
        DATA 0
        DATA 0
        DATA 0
```

6.9.2 Magnetic Tape Resource-Specific I/O

The following sub-opcodes apply to resource-specific magnetic tape I/O:

0A	Read Direct
0C	Write Direct

6.9.2.1 Read Direct. Sub-opcode >0A specifies a Read Direct operation. The Read Direct operation reads a record from the magnetic tape and stores the characters in the specified buffer, two characters per word.

The following fields of the basic supervisor call block apply to a Read Direct operation:

- SVC code — 0
- Return code
- Sub-opcode — >0A
- Logical unit number (LUNO)

- <System flags>
- User flags
- Data buffer address
- Read character count
- <Actual read count>

The system flags defined for all magnetic tape operations apply to a Read Direct operation.

The initiate flag in the user flag field applies to a Read Direct operation:

0	1	2	3–4	5	6	7
↑						

2279554

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

The logical unit number (LUNO) field contains the LUNO assigned to the magnetic tape from which a record is to be read.

The data buffer address is the address of the buffer into which DNOS places the record.

The read character count is the length of the buffer.

DNOS returns the number of characters stored in the buffer in the actual read count field.

The Read Direct operation reads the characters listed in Appendix B for magnetic tape. The operation stores the characters, packed one per byte. The Read operation continues to the end of the record, or until the buffer is full.

When the record has been read, the number of characters read is stored in the actual read count field and the operation terminates. If the number of characters is odd, an additional character is stored in the buffer, but the odd value is stored in the read count field.

When an EOF mark is read, the device service routine (DSR) sets the EOF flag in the system flags byte, stores zero in the actual read count field, and terminates the operation.

The following is an example of the source code for a supervisor call block for a Read Direct operation and code for the read buffer:

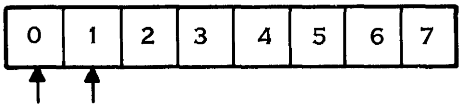
```
RDMT     DATA 0                READ RECORD FROM MAG TAPE ASSIGNED
         BYTE >0A,>3D          TO LUNO >3D IN THE INITIATE I/O
         BYTE 0,>80            MODE.
         DATA RDB
         DATA 150
         DATA 0
         DATA 0
RDB      BSS 150               READ BUFFER
```

6.9.2.2 Write Direct. Sub-opcode >0C specifies a Write Direct operation. The Write Direct operation transfers a record from the specified buffer to the magnetic tape.

The following fields of the basic supervisor call block apply to a Write Direct operation:

- SVC code — 0
- Return code
- Sub-opcode — >0C
- Logical unit number (LUNO)
- <System flags>
- User flags
- Data buffer address
- Write character count

The following system flags apply to a Write Direct operation:

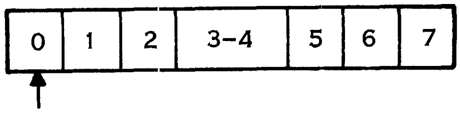


2279555

Bit 0 — Busy flag. Set by system as follows:
1 — Busy.
0 — Operation completed.

Bit 1 — Error flag. Set by system as follows:
1 — Error.
0 — No error.

The following user flags apply to a Write Direct operation:



2279556

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

The logical unit number (LUNO) field contains the LUNO assigned to the magnetic tape to which a record is to be written.

The data buffer address is the address of the buffer that contains the record to be written.

The write character count is the number of characters to be written on the magnetic tape.

The Write Direct operation writes a record on the magnetic tape. The characters in the buffer are not translated.

The following is an example of the source code for a supervisor call block for a Write Direct operation:

```
WAMT    DATA 0              WRITE RECORD TO MAG TAPE ASSIGNED TO
        BYTE >C,>3C         LUNO >3C INITIATE MODE.
        BYTE 0,>80
        DATA WRBUFF
        DATA 0
        DATA 60
```

6.10 CARD READER INPUT

DNOS supports resource-independent and resource-specific input for the card reader.

The following I/O SVC block for card reader operations is the basic block used for all operations. If an extension to this block is necessary for a particular operation, it is indicated in the operation description.

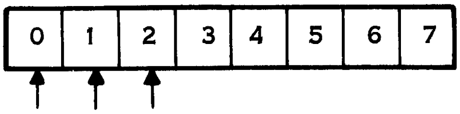
SVC >00 -- I/O OPERATIONS

ALIGN ON WORD BOUNDARY
CAN BE INITIATED AS AN EVENT

DEC	HEX		
0	0	>00	<RETURN CODE>
2	2	SUB-OPCODE	LUNO
4	4	<SYSTEM FLAGS>	USER FLAGS
6	6	DATA BUFFER ADDRESS	
8	8	READ CHARACTER COUNT	
10	A	WRITE CHARACTER COUNT/<ACTUAL READ COUNT>	

2279470

The system flags (byte 4) in the supervisor call block apply to card reader input. These flags are:



2279557

Bit 0 — Busy flag. Set by system as follows:
1 — Busy.
0 — Operation completed.

Bit 1 — Error flag. Set by system as follows:
1 — Error.
0 — No error.

Bit 2 — End-of-file. Set by system as follows:
1 — A card was read with /* punched in columns 1 and 2.
0 — A Read operation read a card that did not have /* in columns 1 and 2.

Two user flags (byte 5) in the supervisor call block apply to card reader input. However, significance of these flags differs for various operations. The flags that apply to each operation are described in the detailed description of each operation.

The operations appropriate for the card reader are described in subsequent paragraphs. The following sub-opcodes, which do not apply to card readers, produce the indicated results:

05	Ignored
06	Ignored
07	Ignored
08	Error
0B	Error
0C	Error
0D	Ignored
0E	Ignored
0F	Ignored

6.10.1 Card Reader Resource-Independent Input

The sub-opcodes for resource-independent card reader input are as follows:

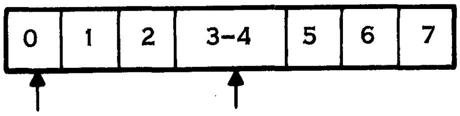
00	Open
01	Close
02	Close, Write EOF
03	Open and Rewind
04	Close and Unload
09	Read ASCII

6.10.1.1 Open. Sub-opcode >00 specifies an Open operation. The Open operation is required for the card reader. However, whether or not DNOS validates the Open operation is specified when the system is generated. Validation does not allow an Open operation when the Open operation would result in a conflict with I/O to the same device by another task.

The following fields of the basic supervisor call block apply to an Open operation:

- SVC code — 0
- Return code
- Sub-opcode — >00
- Logical unit number (LUNO)
- User flags
- Data buffer address
- Read character count

The following flags in the user flag field apply to an Open operation:



2279558

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

Bits 3–4 — Access privilege flag. Set as follows:
00 — Exclusive write.
01 — Exclusive all.
10 — Shared.
11 — Read only.

The logical unit number (LUNO) field contains the LUNO assigned to the card reader to be opened.

The Open operation returns the device type code in the data buffer address field (bytes 6 and 7) of the supervisor call block. The device type code for the card reader is 4.

When the calling task places zero in the read character count field (bytes 8 and 9) of the supervisor call block, the Open operation returns the default logical record length for the device. The default logical record length for the card reader is >50.

The following is an example of the source code for a supervisor call block to open a card reader:

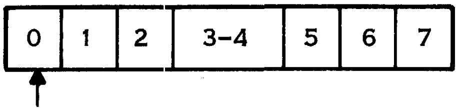
```
OCR     DATA 0              OPEN CARD READER ASSIGNED TO
        BYTE 0,>2A          LUNO >2A.
        DATA 0
CRT     DATA 0
LGT     DATA 0
        DATA 0
```

6.10.1.2 Close. Sub-opcode >01 specifies a Close operation. The Close operation ends I/O to a LUNO from the calling task. The LUNO remains assigned to the device, and can be opened again for additional I/O operations. When a task terminates, DNOS closes all LUNOs that have been opened by the task.

The following fields of the basic supervisor call block apply to a Close operation:

- SVC code — 0
- Return code
- Sub-opcode — >01
- Logical unit number (LUNO)
- User flags

The initiate flag in the user flag field applies to a Close operation:



2279559

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

The logical unit number (LUNO) field contains the LUNO to be closed.

The following is an example of the source code for a supervisor call block to close a card reader:

```
CMT      DATA 0                    CLOSE CARD READER ASSIGNED TO LUNO >2F.
         BYTE 1,>2F
         DATA 0
         DATA 0
         DATA 0
```

6.10.1.3 Close, Write EOF. The Close, Write EOF operation, sub-opcode >02, consists of a Close operation, for the card reader.

6.10.1.4 Open and Rewind. The Open and Rewind operation, sub-opcode >03, is an Open operation, for the card reader.

6.10.1.5 Close and Unload. The Close and Unload operation, sub-opcode >04, consists of a Close operation, for the card reader.

6.10.1.6 Read ASCII. Sub-opcode >09 specifies a Read ASCII operation. The Read ASCII operation reads a record from a punched card and stores the characters in the specified buffer, one character per byte.

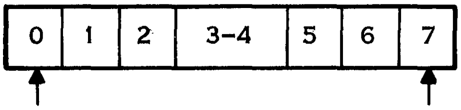
The following fields of the basic supervisor call block apply to a Read ASCII operation:

- SVC code — 0
- Return code
- Sub-opcode — >09
- Logical unit number (LUNO)
- <System flags>
- User flags
- Data buffer address

- Read character count
- <Actual read count>

The system flags defined for all card reader operations apply to a Read ASCII operation.

The following flags in the user flag field apply to a Read ASCII operation:



2279560

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

Bit 7 — Blank adjustment flag. Set as follows:
1 — Read with blank adjustment.
0 — Read without blank adjustment.

The logical unit number (LUNO) field contains the LUNO assigned to the card reader.

The data buffer address is the address of the buffer into which DNOS places the record.

The read character count is the length of the buffer.

DNOS returns the number of characters stored in the buffer in the actual read count field.

The Read ASCII operation recognizes the characters listed in Appendix B for the card reader. The device service routine (DSR) returns an error code when the punches on the card cannot be translated and terminates the input. The operation stores the characters, packed two per word. The Read operation continues until a card has been read; when the Read operation specifies more than 80 characters, the characters on the card are stored in the first 80 bytes of the buffer. When the operation specifies fewer than 80 characters, the remaining characters on the card are not stored in the buffer.

When the record has been read, the number of characters read is stored in the actual read count field and the operation terminates.

When blank adjustment is specified for variable length records, blanks are stored in the buffer to fill the record. That is, when the record length is less than the buffer size, the device service routine (DSR) supplies blanks (>20) to fill the buffer. The character count returned in bytes 10 and 11 includes the blanks supplied by the DSR.

When column 1 of a card contains / and column 2 contains *, the DSR sets the EOF flag in the system flags byte, stores zero in the actual read count field, and terminates the operation.

The following is an example of the source code for a supervisor call block for a Read ASCII operation and code for the read buffer:

```
RDCR    DATA 0          READ RECORD FROM CARD READER ASSIGNED
        BYTE 9,>2D      TO LUNO >2D IN THE INITIATE I/O MODE.
        BYTE 0,>80
        DATA CDB
        DATA 80
        DATA 0
CDB     BSS 80          READ BUFFER
```

6.10.2 Card Reader Resource-Specific Input

The card reader operation specific to the card reader is the Read Direct operation, sub-opcode >0A.

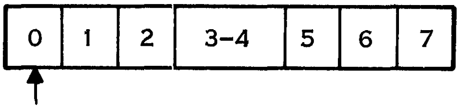
6.10.2.1 Read Direct. Sub-opcode >0A specifies a Read Direct operation. The Read Direct operation reads a punched card and stores the data as binary data, one card column per word.

The following fields of the basic supervisor call block apply to a Read ASCII operation:

- SVC code — 0
- Return code
- Sub-opcode — >0A
- Logical unit number (LUNO)
- <System flags>
- User flags
- Data buffer address
- Read character count
- <Actual read count>

The system flags defined for all card reader operations apply to a Read Direct operation.

The initiate flag in the user flag field applies to a Read Direct operation:



2279561

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

The logical unit number (LUNO) field contains the LUNO assigned to the card reader.

The data buffer address is the address of the buffer into which DNOS places the record.

The read character count is the length of the buffer.

DNOS returns the number of characters stored in the buffer in the actual read count field.

The Read Direct operation reads a card column and stores the data in a word of the buffer. The four most significant bits of the word are set to zero; the holes in the card are stored as ones in card row order (top to bottom), as follows:

BIT:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
CONTAINS:	0	0	0	0						Row						
					12	11	0	1	2	3	4	5	6	7	8	9

2279562

The Read operation continues until a card has been read; when the Read operation specifies more than 160 bytes (80 columns), the data on the card is stored in the 160 low order addresses of the buffer. When the operation specifies fewer than 160 bytes (80 columns), the data in the remaining columns on the card are not stored in the buffer.

When the record has been read, the number of characters read is stored in the actual read count field and the operation terminates. The number of characters read is two counts per word, although each word carries one character (12 bits) in each word.

The Read Direct operation does not recognize an EOF record.

The following is an example of the source code for a supervisor call block for a Read Direct operation and code for the read buffer:

```
RDDCR   DATA 0          READ RECORD FROM CARD READER ASSIGNED
        BYTE 9,>2D      TO LUNO >2D IN THE INITIATE I/O MODE.
        BYTE 0,>80
        DATA CDBF
        DATA 160
        DATA 0
CDBF    BSS 160         READ BUFFER
```

6.11 DIRECT DISK I/O

DNOS supports resource-specific direct disk I/O to all DNOS disks, and to double-sided, double-density diskettes. Direct disk I/O accesses data on the disk by physical address, rather than as a physical record of a file. Because effective use of direct disk I/O requires knowledge of disk organization and of allocation techniques used by DNOS, direct disk I/O operations may only be executed by privileged tasks and system tasks.

Several direct I/O operations require addressing the disk by track. To determine a track address requires a knowledge of the physical organization of data on a disk. Figure 6-5 shows the concept as it applies to a disk pack; a disk with a single platter is organized similarly. Notice that a cylinder includes a recording band on each surface; that is, it includes all the data that could be accessed without repositioning the heads. The recording band on each surface is called a track; in a given head position, each head accesses a track. The track is divided into sectors by timing marks.

The formula for computing track numbers is shown in Figure 6-5. Expressed in symbols the formula is:

$$T = (c \times H) + h$$

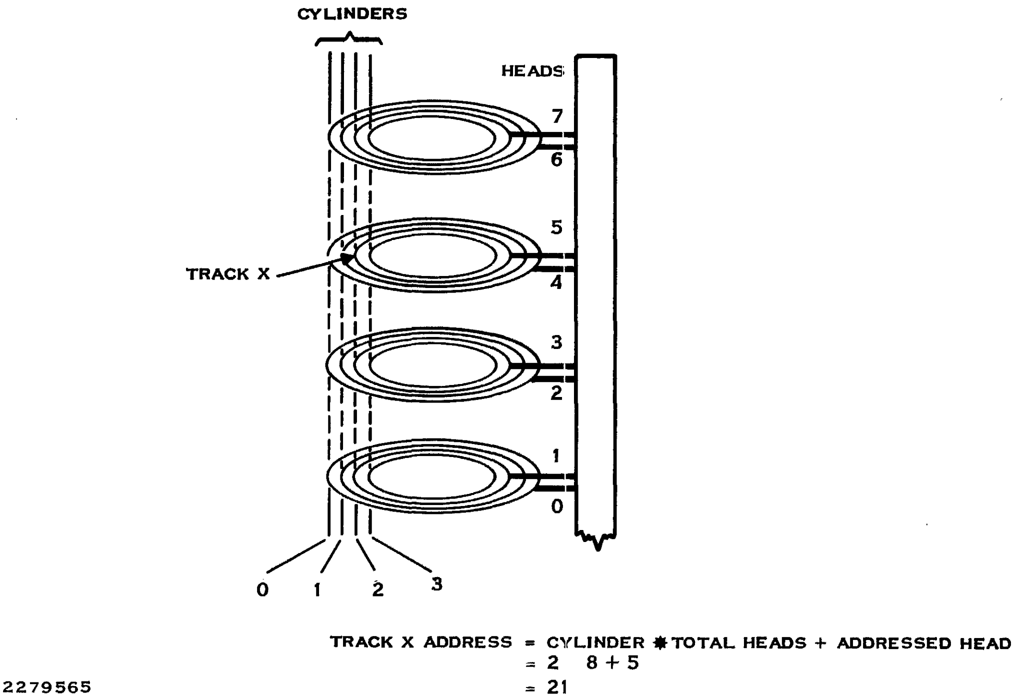


Figure 6-5. Track Addressing

where:

T is the track number.

c is the cylinder number. The number of the cylinder that includes the desired track.

H is the total number of heads (surfaces).

h is the head. The number of the head (surface) that includes the desired track.

For a diskette, the number of heads is two, even-numbered tracks are on surface 0, and odd numbered tracks are on surface 1.

The following I/O SVC block for direct disk operations is the basic block used for all operations. If an extension to this block is necessary for a particular operation, it is indicated in the operation description.

SVC >00 -- I/O OPERATIONS

ALIGN ON WORD BOUNDARY
CAN BE INITIATED AS AN EVENT

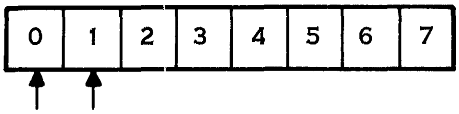
DEC	HEX		
0	0	>00	<RETURN CODE>
2	2	SUB-OPCODE	LUNO
4	4	<SYSTEM FLAGS>	USER FLAGS
6	6	DATA BUFFER ADDRESS	
8	8	READ CHARACTER COUNT	
10	A	WRITE CHARACTER COUNT/ < ACTUAL READ COUNT >	

2279470

The subset of sub-opcodes for direct disk I/O is as follows:

00	Open
01	Close
02	Close, Write EOF
03	Open and Rewind
04	Close, Unload
05	Read Format
08	Write Format
09	Read by ADU
0A	Read by Track
0B	Write by ADU
0C	Write by Track
0E	Store Registers
0F	Read Format
10	Write Deleted Sector
11	Read Deleted Sector
12	Write Format with Interleaving

The system flags (byte 4) in the supervisor call block apply to direct disk I/O. These flags are:



2279563

Bit 0 — Busy flag. Set by system as follows:
1 — Busy.
0 — Operation completed.

Bit 1 — Error flag. Set by system as follows:
1 — Error.
0 — No error.

User flags (byte 5) in the supervisor call block apply to direct disk I/O. However, significance of these flags differs for various operations. The flags that apply to each operation are described in the detailed description of each operation.

The operations appropriate for direct disk I/O are described in subsequent paragraphs. The following sub-opcodes do not apply; they produce the indicated results:

06	Ignored
07	For DNOS use only
0D	Ignored

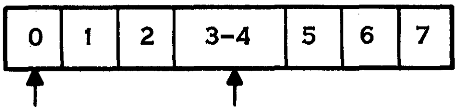
6.11.1 Open

Sub-opcode >00 specifies an Open operation. The Open operation is required for direct disk I/O. However, DNOS does not validate the Open operation; that is, a possible conflict with I/O to the same device by another task is not detected.

The following fields of the basic supervisor call block apply to an Open operation:

- SVC code — 0
- Return code
- Sub-opcode — >00
- Logical unit number (LUNO)
- User flags
- Data buffer address

The following flags in the user flag field apply to an Open operation:



2279564

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

Bits 3-4 — Access privilege flag. Set as follows:
00 — Exclusive write.
01 — Exclusive all.
10 — Shared.
11 — Read only.

The logical unit number (LUNO) field contains the LUNO assigned to the disk.

The Open operation only returns the device type code in the data buffer address field (bytes 6 and 7) of the supervisor call block. The device type code for the disk (and also for the diskette) is 6.

The following is an example of the source code for a supervisor call block to open a disk for direct I/O:

```
OD       DATA 0                   OPEN DISK ASSIGNED TO LUNO >D4.
         BYTE 0,>D4
         DATA 0
D        DATA 0
         DATA 0
         DATA 0
```

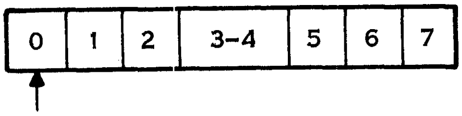
6.11.2 Close

Sub-opcode >01 specifies a Close operation. The Close operation ends I/O to a LUNO from the calling task. The LUNO remains assigned to the device, and may be opened again for additional I/O operations. When a task terminates, DNOS closes all LUNOs that have been opened by the task.

The following fields of the basic supervisor call block apply to a Close operation:

- SVC code — 0
- Return code
- Sub-opcode — >01
- Logical unit number (LUNO)
- User flags

The following user flag applies to a Close operation:



2279569

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

The logical unit number (LUNO) field contains the LUNO to be closed.

The following is an example of the source code for a supervisor call block to close a disk:

```
CVDT      DATA 0                          CLOSE VDT ASSIGNED TO LUNO >24.
          BYTE 1,>24
          DATA 0
          DATA 0
          DATA 0
          DATA 0
```

6.11.3 Close, Write EOF

The Close, Write EOF operation, sub-opcode >02, is identical to the Close operation.

6.11.4 Open and Rewind

The Open and Rewind operation, sub-opcode >03, is identical to the Open operation, for direct disk I/O.

6.11.5 Close and Unload

The Close and Unload operation, sub-opcode >04, is identical to the Close operation.

6.11.6 Read Format

Sub-opcode >05 specifies a Read Format operation. The Read Format operation returns 12 bytes of disk format and track format information for a specified track.

The following fields of the basic supervisor call block apply to a Read Format operation:

- SVC code — 0
- Return code
- Sub-opcode — >05
- Logical unit number (LUNO)
- User flags
- Data buffer address
- <Actual read count>

The following extension to the basic supervisor call block applies to a Read Format operation:

DEC	HEX		
12	C	TRACK ADDRESS	
14	E	SECTORS/RECORD	SECTOR NO.

2279566

The track address field of the supervisor call block extension applies to a Read Format operation. The system flags that apply to direct disk operations apply to a Read Format operation. The following flags in the user flag field apply to a Read Format operation:

0	1	2	3–4	5	6	7
↑						

2279567

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

The logical unit number (LUNO) field contains the LUNO assigned to the disk unit. The system assigns a reserved system LUNO to each disk unit.

The data buffer address is the address of the buffer into which DNOS places the format information. The buffer should contain 12 bytes.

The read character count is the length of the buffer.

DNOS returns the number of characters stored in the buffer in the actual read count field. For a Read Format operation, the system returns >000C in this field.

After a Read Format operation, the data buffer contains twelve bytes of information. The twelve bytes contain the following fields:

DEC	HEX		
0	0	WORDS/TRACK	
2	2	SECTORS/TRACK	OVERHEAD/RECORD
4	4	NO. HEADS	NO. CYLINDERS
6	6	SECTORS/RECORD	RECORDS/TRACK
8	8	WORDS/RECORD	
10	A	INTERLEAVING FACTOR	

2279568

Disk-related data:

Byte	Contents
0–1	The number of words per track on the disk.
2	The number of sectors per track on the disk.
3	The number of words of overhead per record on the disk.
4–5	Bits 0–4 — The number of heads on the disk. Bits 5–15 — The number of cylinders on the disk.

Track-related data:

Byte	Contents
6	The number of sectors per record on the track.
7	The number of records on the track.
8–9	The number of words per record on the track.
10–11	The interleaving factor for the disk. Disks formatted by a Write Format operation have a value of one. Disks formatted by a Write Format with Interleaving operation receive the value supplied when the disk was formatted.

Before the tracks on a disk can be used for data storage, they must be formatted. Formatting, performed when a disk is initialized, defines the physical record length of the records on each track. The Read Format operation reads the format information for a specified track.

The following is an example of the source code for a supervisor call block for a Read Format operation and code for the read buffer:

```
RDFMT    DATA 0                    READ STATUS OF TRACK 35
         BYTE 5,>D3                ON DISK ASSIGNED TO
         DATA 0                    LUNO >D3.
         DATA FT
         DATA 12
FTL      DATA 0
         DATA 35
         DATA 0
FT       BSS 12                    FORMAT BUFFER
```

6.11.7 Write Format

Sub-opcode >08 specifies a Write Format operation. The Write Format operation formats the specified track of the disk.

The following fields of the basic supervisor call block apply to a Write Format operation:

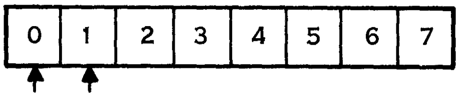
- SVC code — 0
- Return code
- Sub-opcode — >08
- Logical unit number (LUNO)
- <System flags>
- User flags
- Read character count

The Write Format operation requires the extension of the supervisor call block shown for the Read Format operation. The following fields of the extension apply:

- Track address
- <Sectors/record>

The system flags defined for all direct disk operations apply to a Write Format operation.

The following flags in the user flag field apply to a Write Format operation:



2279659

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

The logical unit number (LUNO) field contains the LUNO assigned to the disk.

The read character count field (bytes 8 and 9) contains the physical record length for the track to be formatted.

The track address field contains the address of the track to be formatted.

Formatting, performed when a disk is initialized, defines the physical record length of the records on each track. The Write Format operation formats a single track. Using the physical record length, the system formats the track, and returns the number of sectors per record in byte 14 of the extended supervisor call block.

CAUTION

Reformatting a track destroys any data on the track. A track that is formatted for a number of sectors per record other than one can only be accessed with direct disk I/O.

The following is an example of the source code for a supervisor call block to format track 40 on a disk:

```
FMT     DATA 0              FORMAT TRACK 40 ON DISK ASSIGNED TO
        BYTE >8,>D4         LUNO >D4
        DATA 0
        DATA 0
        DATA 588
        DATA 0
        DATA 40
SR      DATA 0
```

6.11.8 Read by ADU

Sub-opcode >09 specifies a Read by Allocatable Disk Unit (ADU) operation. A Read by ADU operation reads a record from the disk, starting at a specified sector of an ADU.

The following fields of the basic supervisor call block apply to a Read by ADU operation:

- SVC code — 0
- Return code
- Sub-opcode — >09
- Logical unit number (LUNO)
- <System flags>
- User flags
- Data buffer address
- Read character count
- <Actual read count>

The following extension to the basic supervisor call block applies to a Read by ADU operation:

DEC	HEX	
12	C	ADU NUMBER
14	E	SECTOR OFFSET IN ADU

2279570

The following fields of the supervisor call block extension apply to a Read by ADU operation:

- ADU number
- Sector offset in ADU

The system flags defined for all direct disk operations apply to a Read by ADU operation.

The following flags in the user flag field apply to a Read by ADU operation:

0	1	2	3–4	5	6	7
↑						

2279571

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

The logical unit number (LUNO) field contains the LUNO assigned to the disk from which a record is to be read.

The data buffer address is the address of the buffer into which DNOS places the record. The address must be a word address.

The read character count is the length of the buffer. The value must be an even number.

DNOS returns the number of characters stored in the buffer (the number of characters read from the disk) in the actual read count field.

The contents of the ADU number field is the number of the ADU that contains the record to be read.

The sector offset in the ADU field contains the number of a sector relative to the ADU. The Read operation begins at the start of this sector.

The following is an example of the source code for a supervisor call block for a Read by ADU operation and code for the read buffer:

```
RDD     DATA 0             READ RECORD FROM DISK ASSIGNED TO
        BYTE 9,>D5         LUNO >D5 IN THE INITIATE I/O MODE
        BYTE 0,>80
        DATA DRB
        DATA 588
        DATA 0
        DATA 25
        DATA 0
DRB     BSS 588            READ BUFFER
```

6.11.9 Read by Track

Sub-opcode >0A specifies a Read by Track operation. The Read by Track operation is similar to a Read by ADU operation. However, the record to be read is addressed by track and sector number rather than by ADU and sector. Also, flags in the user flag field select options required for surface analysis.

The track and sector address is supplied in the following extension to the supervisor call block:

DEC	HEX		
12	C	TRACK ADDRESS	
14	E	SECTORS/RECORD	SECTOR NO.

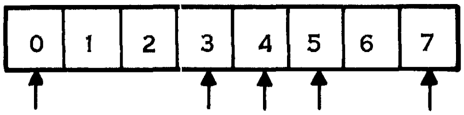
2279572

The track address field contains the address of the track for the Read operation.

The sectors/record field contains the number of sectors per record on the addressed track.

The sector number field contains the number of the sector to be read.

The following flags in the user flag field apply to a Read by Track operation:



2279573

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

Bit 3 — Head offset flag. Set as follows:
1 — Read with head offset.
0 — Read with no head offset.

Bit 4 — Head offset direction flag (valid only when bit 3 is set to one). Set as follows:
1 — Offset head forward.
0 — Offset head backward.

Bit 5 — Transfer inhibit flag. Set as follows:
1 — Inhibit transfer of data.
0 — Transfer data.

Bit 7 — Retry flag. Set as follows:
1 — Do not retry on error.
0 — Retry on error.

The user flags that apply to the Read by Track operation only specify options used in surface analysis. A head offset provided by the disk drive may be applied in either a forward or backward direction under control of bits 3 and 4. A track may be read for detecting errors without actually storing the data from the disk when bit 5 is set to one. Retries of the Read operation in case of error are controlled by bit 7.

The following is an example of the source code for a supervisor call block for a Read by Track operation and code for the read buffer:

```
RDTS    DATA 0               READ RECORD FROM DISK ASSIGNED TO
        BYTE >A,>D5          LUNO >D5 IN THE INITIATE I/O MODE
        BYTE 0,>80
        DATA DRBF
        DATA 588
        DATA 0
        DATA 40
        BYTE 1
        BYTE 24
DRBF    BSS 588              READ BUFFER
```

6.11.10 Write by ADU

Sub-opcode >0B specifies a Write by ADU operation. The Write by ADU operation writes a record on a disk, starting at a specified sector of an ADU.

The following fields of the basic supervisor call block apply to a Write by ADU operation:

- SVC code — 0
- Return code
- Sub-opcode — >0B
- Logical unit number (LUNO)
- <System flags>
- User flags

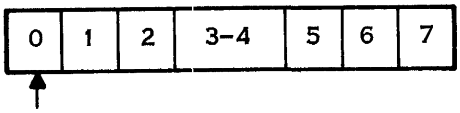
- Data buffer address
- Write character count

The extension to the supervisor call block shown for the Read by ADU operation also applies to the Write by ADU operation. All fields of the extension apply, as follows:

- ADU number
- Sector offset in ADU

The system flags defined for all direct disk operations apply to a Write by ADU operation.

The following flags in the user flag field apply to a Write by ADU operation:



2279574

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

The logical unit number (LUNO) field contains the LUNO assigned to the disk.

The data buffer address is the address of the buffer which contains the record to be written. The address must be a word address.

The write character count is the number of characters to be written on the disk. The value must be an even number.

The contents of the ADU number field is the number of the ADU that contains the record to be read.

The sector offset in ADU field contains the number of a sector relative to the ADU. The Write operation begins at the start of this sector.

When the number of characters written does not fill an integral number of sectors, the system writes zeros in the remaining bytes in the partially-written sector.

The following is an example of the source code for a supervisor call block for a Write by ADU operation:

```
WDA       DATA 0                  WRITE RECORD TO DISK ASSIGNED TO
          BYTE >B,>D3             LUNO >D3 INITIATE MODE.
          BYTE 0,>80
          DATA WRBUFF
          DATA 0
          DATA 588
          DATA 34
          DATA 0
```

6.11.11 Write by Track

The Write by Track operation (sub-opcode >0C) is similar to the Write by ADU operation. The difference is that the record to be read is addressed by track and sector number rather than by ADU and sector.

The track and sector address is supplied in the extension to the supervisor call block shown for the Read by Track operation.

The track address field contains the address of the track for the Write operation.

The sectors/record field contains the number of sectors per record on the addressed track.

The sector number field contains the number of the sector to be written.

The following is an example of the source code for a supervisor call block for a Write by Track operation:

```
WDT       DATA 0                  WRITE RECORD TO DISK ASSIGNED TO
          BYTE >C,>D5             LUNO >D5 IN THE INITIATE I/O
          BYTE 0,>80              MODE.
          DATA WRBUFF
          DATA 0
          DATA 588
          DATA 75
          BYTE 1
          BYTE 12
```

6.11.12 Store Registers

Sub-opcode >0E specifies a Store Registers operation. The Store Registers operation transfers disk parameters from registers in the disk controller to a specified buffer.

The following fields of the basic supervisor call block apply to a Store Registers operation:

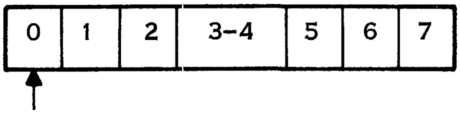
- SVC code — 0
- Return code
- Sub-opcode — >0E

- Logical unit number (LUNO)
- User flags
- Data buffer address
- <Actual read count>

The extension to the basic supervisor call block shown for a Read Format operation applies also to a Store Registers operation. However, all of the fields in the extension are ignored.

The system flags that apply to direct disk operations apply to a Store Registers operation.

The following flags in the user flag field apply to a Store Registers operation:



2279575

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

The logical unit number (LUNO) field contains the LUNO assigned to the disk unit.

The data buffer address is the address of the buffer into which DNOS places the disk information. The buffer should contain six bytes, beginning on a word address.

The read character count is the length of the buffer.

DNOS returns the number of characters stored in the buffer in the actual read count field. The system returns >0006 in this field.

After a Store Registers operation, the data buffer contains six bytes of information, organized in the following fields:

DEC	HEX		
0	0	WORDS/TRACK	
2	2	SECTORS/TRACK	OVERHEAD/RECORD
4	4	NO. HEADS	NO. CYLINDER

2279576

Byte	Contents
0–1	The number of words per track on the disk.
2	The number of sectors per track on the disk.
3	The number of words of overhead per record on the disk.
4–5	Bits 0–4 — The number of heads on the disk. Bits 5–15 — The number of cylinders on the disk.

The following is an example of the source code for a supervisor call block for a Store Registers operation and code for the read buffer:

```
SRDC    DATA 0                          STORE REGISTERS FOR DISK
        BYTE >E,>D3                     ASSIGNED TO LUNO >D3
        DATA 0
        DATA REGS
        DATA 6
RL      DATA 0
        DATA 0
        DATA 0
REGS    BSS 6                           REGISTER BUFFER
```

6.11.13 Read Format

Sub-opcode >0F is an alternate sub-opcode for the Read Format operation previously described for sub-opcode >05.

6.11.14 Write Deleted Sector

Sub-opcode >10 specifies a Write Deleted Sector operation. The Write Deleted Sector operation writes the deleted sector data pattern on a diskette sector. The sector is specified with a track and sector address.

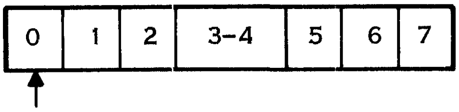
The following fields of the basic supervisor call block apply to a Write Deleted Sector operation:

- SVC code — 0
- Return code
- Sub-opcode — >10
- Logical unit number (LUNO)
- User flags

The track and sector address is supplied in the extension to the supervisor call block shown for the Read by Track operation.

The system flags defined for all direct disk operations apply to a Write Deleted Sector operation.

The following flags in the user flag field apply to a Write Deleted Sector operation:



2279577

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

The logical unit number (LUNO) field contains the LUNO assigned to the diskette.

The track address field contains the address of the track for the Write Deleted Sector operation.

The sectors/record field contains the number of sectors per record on the addressed track.

The sector number field contains the number of the sector to be written.

The standards for diskettes define a data pattern that identifies a deleted sector. The Write Deleted Sector operation writes this data pattern on the specified sector. A sector that has been written as a deleted sector using this sub-opcode can only be read by a Read Deleted Sector operation.

The following is an example of the source code for a supervisor call block to write the deleted sector data pattern on the diskette:

```
WDS    DATA 0                   WRITE DELETED SECTOR ON DISKETTE
       BYTE >10,>D2             ASSIGNED TO LUNO >D2
       DATA 0
       DATA BUFFER
       DATA 0
       DATA 0
       DATA 35
       BYTE 1
       BYTE 4
```

6.11.15 Read Deleted Sector

Sub-opcode >11 specifies a Read Deleted Sector operation. The Read Deleted Sector operation reads a sector of a diskette on which the deleted sector data pattern has been written.

The following fields of the basic supervisor call block apply to a Read Deleted Sector operation:

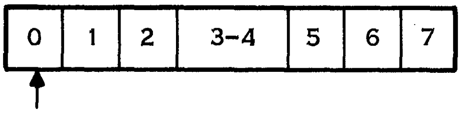
- SVC code — 0
- Return code
- Sub-opcode — >11
- Logical unit number (LUNO)

- <System flags>
- User flags
- Data buffer address
- Read character count
- <Actual read count>

The track and sector address is supplied in the extension to the supervisor call block shown for the Read Deleted Sector operation.

The system flags defined for all direct disk operations apply to a Read Deleted Sector operation.

The following flags in the user flag field apply to a Read Deleted Sector operation:



2279578

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

The logical unit number (LUNO) field contains the LUNO assigned to the diskette.

The data buffer address is the address of a buffer into which the deleted sector is read. The buffer address must be a word address.

The read character count is the number of characters in the buffer. Since a deleted sector contains a special data pattern, it is appropriate to read only a part of the data. The read character count must be an even number.

The system returns the number of characters transferred to the buffer in the actual read count field.

The track address field contains the address of the track for the Read Deleted Sector operation.

The sectors/record field contains the number of sectors per record on the addressed track.

The sector number field contains the number of the sector to be read.

A Read Deleted Sector operation is the only operation that reads a deleted sector successfully. When a Read Deleted Sector operation addresses a sector that has not been deleted, the operation returns an error code and no data is transferred to the data buffer.

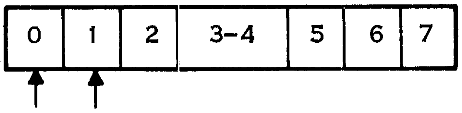
The following is an example of the source code for a supervisor call block for a Read Deleted Sector operation and code for the read buffer:

```
RDDS      DATA 0               READ DELETED SECTOR FROM DISKETTE
          BYTE >11,>D5         ASSIGNED TO LUNO >D5 IN THE
          BYTE 0,>80           INITIATE I/O MODE.
          DATA DSB
          DATA 10
          DATA 0
          DATA 32
          BYTE 1
          BYTE 12
DSB       BSS 10               READ BUFFER
```

6.11.16 Write Format with Interleaving

Sub-opcode >12 specifies a Write Format with Interleaving operation. The Write Format with Interleaving operation specifies an interleaving factor for the diskette. The operation may specify the data pattern used in formatting the diskette. The operation is otherwise identical to the Write Format operation, sub-opcode >08.

The following flags in the user flag field apply to a Write Format with Interleaving operation:



2279579

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

Bit 1 — Format data location flag. Set as follows:
1 — Data in buffer.
0 — Data in call block.

When the format data location flag is set to 1, a two-word buffer with the following contents is required:

DEC	HEX	
0	0	INITIALIZATION WORD
2	2	INTERLEAVING FACTOR

2279580

The initialization word is a word that the system writes on the diskette during formatting. The interleaving factor should be 1 for diskettes used with DNOS. For a diskette that is to be used in a DS990 Model 1 system (DX5 or TX5), the interleaving factor should be 3. The address of this buffer is placed in the data buffer address field (bytes 6 and 7) of the supervisor call block.

When the format data location flag is set to zero, the write character count field (bytes 10 and 11) of the supervisor call block contains the interleaving factor. An initialization word supplied by the system is written on the diskette.

Interleaving arranges the sectors on the diskette to allow time for transfer of data between operations that address the sectors in sequence. An interleaving factor of 1 arranges the sectors in sequence; a factor of 2 interleaves a sector between sectors 0 and 1, and between sectors 1 and 2, and so on. The value of 3 means that two sectors pass the heads between consecutively-numbered sectors. This allows the period of time required to write two sectors for the transfer of data.

The following is an example of the source code for a supervisor call block to format track 40 on a diskette with an interleaving factor:

```
FMTI     DATA  0             FORMAT TRACK 40 ON DISKETTE
         BYTE  >12,>D4       ASSIGNED TO LUNO >D4 WITH
         DATA  0             INTERLEAVING FACTOR OF 3
         DATA  0
         DATA  588
         DATA  3
         DATA  40
SRI      DATA  0
```

6.12 DUMMY DEVICE I/O

The dummy device can be assigned to a LUNO to effectively skip all I/O operations to that LUNO. The device name is DUMY. The following operations to the dummy device are supported:

- Open
- Open and Rewind
- Read ASCII
- Read Direct

The Open and Open Rewind operations to the dummy device return device type 0 in the data buffer address field (bytes 6 and 7) of the supervisor call block. The default logical record length returned in bytes 8 and 9 is zero. The Read ASCII and Read Direct operations set the EOF flag.

The following sub-opcodes produce the indicated results:

01	Ignored
02	Ignored
04	Ignored
05	Ignored
06	Ignored
07	Ignored
08	Ignored
0B	Ignored
0C	Ignored
0D	Ignored
0E	Ignored
0F	Ignored

7

File I/O

7.1 DNOS FILES

File I/O combines the support of the file management and I/O capabilities of DNOS for three types of disk files. The file types are:

- Sequential files
- Relative record files
- Key indexed files

A sequential file consists of records written in the sequence in which they are presented to file management and read in the sequence in which they were written. Resource-independent I/O concepts apply to sequential files; that is, input from a card reader may be substituted for input from a sequential file of card-image records by changing the LUNO assignment.

Relative record files consist of records that are uniquely identifiable by the position they occupy within the file. When a record is written, it is written to a specific position within the file. The position is identified by an integer. Position 0 identifies the first record of the file, position 1 identifies the second record, and so on. A read operation accesses the record by its record number. Relative record files may be written or read in numeric sequence using the number identifying the position of the record. Records may be read or written in random sequence by specifying the record number explicitly.

Three special usage types of relative record files are supported by DNOS. The special usage types are:

- Directory files — Contain information necessary to locate other files and to describe the characteristics of those files.
- Program files — Store the linked object code of task segments, procedure segments, program segments, and overlays in a format that can be executed.
- Image files — Contain the memory image code of programs stored so that the logical record size equals the physical record size which equals the disk sector size.

Key indexed files consist of records that contain one or more keys by which they are accessed. They provide the capability of accessing records by content instead of by position.

Several characteristics of file I/O apply to two or three of these file types. These subjects are discussed in subsequent paragraphs, preceding the descriptions of File I/O Utility operations and I/O operations to each type of file.

7.1.1 Record Blocking

To reduce the number of disk transfers, it is often best to choose physical records large enough to contain several logical records. The size of a physical record should be a multiple or submultiple of the size of an allocatable disk unit (ADU). The technique of storing several logical records in a physical record is called blocking.

When a task first issues a read request, the system actually reads an entire physical record or ADU (whichever is larger) into memory. The physical record is stored in an area of memory called a blocking buffer. Only the part that corresponds to the requested logical record is passed to the requesting task. Subsequent Read and Write operations do not cause immediate disk access but instead refer to the record image in memory (unless the immediate write mode is in effect). The system keeps an accessed physical record (which usually contains several logical records) in memory until the memory area is needed for other purposes. The blocking of logical records into physical records, together with the deferred write capabilities, may substantially improve system throughput. This is especially true for sequential files.

The system always processes sequential files as blocked files. For relative record files blocking only applies when the physical record size specified is at least twice the logical record length.

7.1.2 Blank Adjustment/Compression

Blank compression conserves space on the disk that otherwise would contain blanks. Blank compression is requested for a file when it is created. It is discussed further in the paragraph on the Create File operation.

Blank adjustment is also available for saving disk space on sequential files. Blank adjustment may be specified for Write and Read operations. When blank adjustment is specified for a Write operation, trailing blanks in each record are not written. When blank adjustment is specified for a Read operation, the buffer is filled with blanks following the last character read.

7.1.3 File and Record Protection Features

The system provides the following features for protecting files from program flaws that might otherwise destroy valuable data:

- Delete and write protection
- Record locking

.1.3.1 Delete and Write Protection. The delete and write protection file attributes may be modiied by utility I/O calls described in subsequent paragraphs. Files are created initially without proection; a subsequent call must be made to apply protection. If security was selected during ystem generation (answering YES to the SECURITY? prompt), delete and write protect access is equired to perform these operations.

n attempt to write to or delete a file with write protection will fail and return an error code. An ttempt to delete a file with delete protection will fail and return an error code. These protective ttributes are not intended for file security. (There are nonprivileged supervisor calls to remove rotection.) They do provide protection against program flaws and user errors that might othervise destroy valuable data.

7.1.3.2 Record Locking. Record locking means that although access to a given file may be shared among several users, individual records may be locked to provide exclusive (single user) read and write access. This feature is useful in ensuring that record updates occur one at a time. For example, inventory files might be accessible from several terminals. Record locking can prevent simultaneous updating of a record by two or more users, causing an undetected loss of one of the updates.

EXAMPLE

Without Record Lock

1. User A reads a record.
2. User B reads the same record.
3. User A updates his copy of the record and writes the updated record to disk.
4. User B updates his copy of the record and writes the updated record to disk.

Final Result: User B's update is incorporated in the record, but User A's update is not included.

With Record Lock

1. User A reads a record and locks it by setting the lock/unlock flag (byte 5, bit 5) in the call block.
2. User B attempts to read the same record but must wait for User A to unlock the record.
3. User A updates the record and writes it back to disk, which unlocks it if the lock/unlock flag is set.
4. User B reads the record and locks it.
5. User B updates the record, writes it back to disk, and unlocks it.

Final Result: Both updates are now included in the record.

Record locking requires an entry in memory for each locked record, which imposes a limitation on the number of records that can be locked at one time. The limit depends on the size of memory and usage of memory by other system functions. It is on the order of 1000 records, and an error code is returned when no more memory is available for locked record entries.

7.1.4 Temporary Files

DNOS supports task-temporary and job-temporary files, which are automatically deleted when the task or job terminates.

A task-temporary file is a file used only within the scope of a task. When the temporary flag in the supervisor call block is set to one, an Assign LUNO or Create File I/O Operation can create a task-temporary file. When an Assign LUNO Operation creates (autocreates) a task-temporary file, the system generates the pathname. The name of the disk on which the file will be created can be specified; if no pathname is specified, the file is created on the system disk. A file named by the system is deleted when the LUNO is released. A file named by the user is deleted with the release of the LUNO (or the last of several LUNOs) assigned to it.

A job-temporary file is a temporary file that is required and used only within the scope of a job. These temporary files are local to the job to which the task that creates them is associated, and they are deleted along with the job. A job-temporary file may be accessed by any task within the job.

Using the Assign Logical Name (ALN) SCI command and specifying a resource type of TEMPORARY creates a job temporary file. If the CREATE NOW option is chosen, the file is created immediately; otherwise, the file is created the first time a LUNO is assigned to the logical name.

Access to the file is by logical name. The logical name operations are described in Section 5.

Job-temporary files are deleted when the job terminates or when the logical name is released.

7.1.5 Concatenated and Multifile Sets

Sequential and relative record files may be logically concatenated by setting the values of a logical name to the pathnames of a set of files. Logical concatenation allows access to the set of files, in sequence, without physically concatenating the files. (When required, physical concatenation may be performed by the Copy/Concatenate (CC) SCI command.) A multifile set is a set of key indexed files, the pathnames of which are the values of a logical name. The files in the set are associated in a nonreversible manner. Individual components of concatenated and multifile sets may be on separate disks.

Several restrictions apply to the concatenation of files. The files must be the same type and may not be special-use files (directories, program files, key indexed files, or image files). Relative record files to be concatenated must have the same logical record size. A concatenation of files may not contain both blocked and unblocked records, and any LUNO assigned to a file must be released before concatenating the file. A file may not be concatenated or associated with itself.

Special rules apply to the combining of key indexed files in a multifile set. At the first definition of the multifile set, all but the first file must be empty; none may be a member of an existing multifile set; they must all have the same physical record size; and they must have the same key definitions. In subsequent definitions of these sets, the same files must be associated in the same order, and none of the original set may be omitted. One empty file may be added at the end (but not at any other position). Key indexed files of a multifile set can only be individually accessed as an unblocked file.

The intended use of a multifile set of key indexed files is to permit a larger key indexed file than one disk can store. When a key indexed file can no longer expand because there is insufficient space on the disk, a new file can be created on another disk. By using a logical name, the two files can be used as one. The second file is used as an extension of the first. If the first file contains 5000 physical records, when physical record 5001 is required, the first physical record of the second file, record 0, is used.

Only a few of the file utility operations of the I/O Operations SVC apply to concatenated and multivolume sets, as follows:

91 — Assign LUNO
93 — Release LUNO
99 — Verify Pathname

The Assign Logical Name (ALN) SCI command associates files collectively with a logical name. Actual logical concatenation or creation of a multifile set occurs when a LUNO is assigned to the logical name. A concatenated file may be accessed only for the duration of the logical name. The user must specify the files in the concatenation order desired. Files can be specified by pathname, synonym, logical name or a logical name and pathname combination. However, all forms must resolve to valid pathnames. All files in the concatenation or multifile set must be precreated and online when the logical name is used.

The last file in a concatenation set can be expandable. All other files become nonexpandable until the logical name is released or the job terminates.

When a single end-of-file mark appears at the end-of-medium, the end-of-file is masked. This allows concatenated files to be accessed logically as a single file without the hindrance of intermediate end-of-file marks being returned. Note that any intermediate end-of-file mark not at the end-of-medium is always returned. If two end-of-file marks are encountered at the end-of-medium, a single end-of-file is returned.

Several users can access the same concatenated or multifile set if the access privileges permit. Two concatenated files are identical when they consist of the same pathnames in the same order. To maintain file integrity, an error is returned if any of the precreated files of a concatenated file are being accessed independently. A concatenated file is deleted by deleting the individual files.

7.1.6 End-of-File

An end-of-file (EOF) is a logical position within a relative record or key indexed file and an actual record within sequential files which, when read, causes the EOF status bit to be set. No data is transferred. The EOF status bit is bit 2 of the system flags. Relative record files have one EOF that corresponds to the record following the highest-numbered written record. Sequential files may have more than one EOF. A sequential file is analogous to a reel of magnetic tape, which could contain several files, separated by EOFs on the tape. Thus, a sequential file may consist of multiple data sets, or subfiles, delimited by EOFs. A key indexed file has a logical EOF that corresponds to the record following the record with the largest primary key. For a key indexed file, the EOF applies only to a Read ASCII operation and a Forward Space operation, which access the file sequentially in primary key order.

The internal representation of an EOF in a sequential file is a record of zero length; it is written by a Write EOF operation or by a Close, Write EOF operation. Writing an EOF does not prevent writing more records to the file.

7.2 FILE UTILITY OPERATIONS

A file may be accessed by a pathname or by a logical name, through assignment of a LUNO. The utility functions required for file I/O include the Assign LUNO and Release LUNO operations, and functions to create a file, delete a file, verify or change the pathname, apply or change protection, and add or delete an alias.

7.2.1 Performing Utility Functions

A subset of the sub-opcodes of the I/O Operations SVC (opcode 00) performs I/O utility functions that support file I/O. These I/O utility functions allow a program to:

- Create a file
- Delete a file
- Assign a LUNO
- Release a LUNO
- Assign a new pathname
- Verify a pathname
- Apply write protection
- Apply delete protection
- Remove protection
- Add an alias to a file
- Delete an alias of a file
- Specify the write mode

Only the following operations apply to concatenated files:

- Assign a LUNO
- Release a LUNO
- Verify a pathname

Many of these utility operations require pathnames. The pathname of a file consists of a volume name (which may be implied if it is the system disk), directory names (if any), and a filename. The names within the pathname are separated by periods (.). When the volume name is that of the system disk, it may be omitted. The pathname begins with a period in this case. The number of directory names in the pathname depends upon the organization of the disk. The volume directory and directories at all levels may contain both directories and files. The maximum length of a pathname is 48 characters.

The capability of creating and deleting files with an SVC simplifies execution of the program by creating files that otherwise would have to be created by the user prior to executing the program. Similarly, the assignment of LUNOs and the release of LUNO assignments by the program makes proper execution of the program less dependent on the user.

A program can also change the pathname of a file. A change in a pathname may change either the filename or one or more directory names, but not the volume name.

A program can verify a pathname. This consists of verifying the syntax of a pathname and also verifying the existence of the file corresponding to the pathname.

Files are created without protection. A file may be protected from being deleted, or from being written to, by a utility function of the I/O Operations SVC. Also, protection may be removed using another function.

An alias is an alternate name for a directory or file, making a file accessible using a pathname with an alias instead of the directory name or filename. An alias may be used to avoid recoding and reassembling a program to change a pathname when the actual pathname is no longer the pathname that appears in the code. A program may define an alias or delete an alias.

DNOS normally defers the actual writing of a record in a file until the memory occupied by the record is required for other purposes or until the file is closed. Any write error that occurs when the record is written may thus be delayed beyond the point at which the program tests for a write error. DNOS supports an immediate write mode that avoids the problem by actually writing the record in the file when the write SVC is executed. The immediate write mode should also be used for files that must be updated without the delay that could otherwise occur. For example, the system log must be written in the immediate write mode to prevent loss of messages when the system fails. The immediate write mode is specified when a file is created or whenever the specify write mode utility function is performed. The deferred write mode may also be specified when the file is created or with the same utility function.

Specifying parameters adds functions to many of the I/O operations. The parameter address field of the request block points to a parameter list. The parameter list can contain multiple parameters. Parameters that apply only to a specific I/O operation are discussed in the description of that operation. The following discussion applies to those parameters that may apply to multiple I/O operations.

In a secure environment, it may be necessary to issue an I/O utility operation specifying the rights of a user other than the user issuing the call; this is useful in server jobs that process requests from other jobs. By issuing the operation with a user ID parameter, the servor job can assume the security of the requestor. In a secure environment, the following I/O utility operations can specify a user ID as an SVC I/O call block parameter:

- Assign LUNO — This command assigns a LUNO if the specified user has any access rights to the file. DNOS verifies the user's access rights for all subsequent I/O operations that use the assigned LUNO.

- Create File — This command creates a file with full access rights given to the specified user's creation access group.

- Delete file — This command deletes a file if the specified user ID has delete access to the file.

- Unprotect file — This command removes write and delete protection from a file if the specified user ID has write and delete access to the file.

- Write protect file — This command write protects a file if the specified user ID has write and delete access to the file.

- Delete Protect File — This command delete protects a file if the specified user ID has write and delete access to the file.

f specified, the user ID must be the first entry in the parameter list. The user ID and user passcode ields are eight bytes long; if the specified user ID or user passcode is less than eight characters, he user ID or user passcode must be left-justified and the field right-filled with blanks. For secu-ity bypass tasks, the passcode does not need to be specified. The parameter list address must be laced in the SVC I/O call block, and the utility flag bit must be set. The following diagram shows a ser ID and passcode as the only entry in the parameter list.

>13	>0	TOTAL LENGTH OF LIST— LENGTH BYTE/RESERVED
>02	>10	SUBLIST NUMBER/LENGTH OF USER ID + PASSCODE
USER ID		8 BYTES
PASSCODE		8 BYTES

2285030

Byte	Contents
0	Total parameter list minus length byte in bytes
1	Reserved — Must be set to zero.
2	>02 — User ID sublist number

3 Length of user ID sublist in bytes

4–11 User ID — Left-justified and right-filled with blanks

12–19 Passcode — Unencrypted, left-justified and right-filled with blanks

The utility operations require an extended supervisor call block, described in Section 5 and displayed below.

SVC >00 -- I/O OPERATIONS
(UTILITY SUB-OPCODE)

ALIGN ON WORD BOUNDARY
CAN BE INITIATED AS AN
EVENT

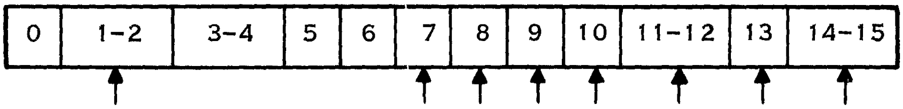
DEC	HEX		
0	0	>00	<RETURN CODE>
2	2	SUB-OPCODE	LUNO
4	4	<SYSTEM FLAGS>	USER FLAGS
6	6	<RESOURCE TYPE>	
8	8	RESERVED	
10	A		
12	C	KEY DEF. BLOCK ADDR/DEF. PHYS. REC. SIZE	
14	E	RESERVED	
16	10	UTILITY FLAGS	
18	12	DEFINED LOGICAL RECORD LENGTH	
20	14	DEFINED PHYSICAL RECORD LENGTH	
22	16	PATHNAME ADDRESS	
24	18	PARAMETER ADDRESS	
26	1A	RESERVED	
28	1C	INITIAL FILE ALLOCATION	
30	1E		
32	20	SECONDARY FILE ALLOCATION	
34	22		

2279581

7.2.1.1 Creating Files. To create a file, a program executes an I/O Operations SVC with sub-opcode >90. The following fields of the utility supervisor call block apply:

- SVC code — 0
- Return code
- Utility sub-opcode — >90
- Key definition block address (key indexed files)
- Default physical record size (directory files)
- Numbers of tasks, procedures, and overlays (program files)
- Utility flags
- Logical record length
- Physical record length
- Pathname address
- Parameter address
- Initial file allocation
- Number of directory entries (directory files)
- Secondary file allocation

The following utility flags (bytes 16 and 17 of the call block) apply:



2279582

Bits 1-2 — File usage flag. Set as follows:
00 — No special usage.
01 — Directory file.
10 — Program file.
11 — Image file.

Bit 7 — Parameter flag. Set as follows:
1 — Parameters are present and pointed to by bytes >18 and >19.
0 — No parameters.

Bit 8 — LRL flag. Set as follows:

1 — Place the logical record length in bytes >12 and >13.

0 — Place the logical record length in bytes >8 and >9. It is recommended that you set the LRL flag to 1 and place the logical record length in bytes 12 and 13.

Bit 9 — Temporary file flag. Set as follows:

1 — Temporary file.

0 — Not a temporary file.

The temporary file will exist until the last LUNO assigned to the file is released.

Bit 10 — Write mode flag. Set as follows:

1 — Immediate write mode.

0 — Deferred write mode (normal mode).

The setting of the write mode flag does not apply to a key indexed file, which is always created in the immediate write mode.

Bits 11-12 — Data format flag. Set as follows:

00 — Normal record image.

01 — Blank compressed.

The setting of the data format flag does not apply to a key indexed file, which is always blank compressed.

Bit 13 — Allocation flag. Set as follows:

1 — Expandable file.

0 — Fixed size file.

Bits 14-15 — File type flag. Set as follows:

01 — Sequential file.

10 — Relative record file.

11 — Key indexed file.

A file consists of a set of data structures called logical records. Division of a file into logical records does not necessarily correspond to the physical division of data on the disk. That is, there are two types of records: logical records, the data structures read and written by programs, and physical records, the data structures actually transferred to and from the disk. Typically, the physical record contains several logical records. Both logical and physical record sizes must be specified, explicitly or by default, when a file is created.

The type of the file is specified when the file is created:

- Sequential
- Relative record
- Key indexed

Within the relative record file type are three special use categories:

- Directory files
- Program files
- Image files

When creating a relative record file, the user must specify the special use category or specify that no special use category applies.

The pathname of a file is assigned when the file is created. The ASCII characters of the pathname are placed in an area of memory that is preceded by a byte that contains the number of characters in the pathname. The address of the byte that contains the character count is placed in the supervisor call block as the pathname address.

The Performing Utility Function paragraph of this section shows how to specify user ID parameters in a secure environment. This option allows the creation of a file with full access rights given to the user's creation access group. This option does not apply to directories.

The initial file allocation applies to all files. DNOS files may be expandable or of fixed length. The initial allocation determines the number of logical records allocated to the file initially, in the case of an expandable file, or the total allocation, in logical records, for a fixed length file. The allocation flag is set to one when the file is expandable, and the secondary file allocation field contains the number of additional logical records requested when the initial allocation has been filled. Additional secondary allocations are made, as required.

Additional disk space is allocated contiguous to space already occupied by the file, when possible. When the initial allocation is filled, additional space is requested. When the requested space is available contiguous to the initial allocation, the initial allocation is extended by the amount of the request. When contiguous space is available (but not as much as is requested), the initial allocation is extended to occupy the available contiguous space. Only when no contiguous space is available is a secondary allocation made. The size of the secondary allocation is the requested size or the largest contiguous block of disk space, whichever is smaller.

The amount of disk space requested is the larger of two values, each computed by a formula. The first formula is based on the number of secondary allocations, as follows:

A = SA × (2**n)

where:

A is the amount of disk space requested.

SA is the secondary allocation field contents (converted to ADUs).

n is the number of secondary allocations, 0 through 15.

The other formula is based on the number of allocations (including extensions to an existing allocation and secondary allocations):

A = TIMTBL(x)

where:

A is the number of ADUs of disk space requested.

x is the number of allocations, initialized to the number of secondary allocations when the LUNO is assigned and incremented for each allocation.

TIMTBL is a table of numbers of physical records.

Table TIMTBL contains the following values:

TIMTBL(0) = 1 physical record
TIMTBL(1) = 2 physical records
TIMTBL(2) = 4 physical records
TIMTBL(3) = 8 physical records
TIMTBL(4) = 12 physical records
TIMTBL(5) = 16 physical records
TIMTBL(6) = 20 physical records
TIMTBL(7) = 24 physical records

The value of TIMTBL(7) also applies to values of x greater than 7.

Because of the conversion of SA in the first formula to ADUs, that formula applies when a physical record is larger than an ADU.

A temporary file is a file that is deleted automatically. After the file is created, one or more LUNOs may be assigned to the file. When the last LUNO assigned to the file is released, DNOS deletes the file.

Normally, a physical record remains in memory until the memory it occupies is required for other purposes. Read and Write operations transfer data from and to the record in memory, but the actual writing of the data to the disk occurs some time after the write SVC is completed. This is a very efficient way to manage a file, since memory accesses are much faster than disk accesses. However, should an error occur when the physical record is actually written, the error code is returned at completion of the next SVC to the LUNO, and the error may be misinterpreted or ignored. The immediate write mode, which applies when the write mode flag is set, forces DNOS to write the physical record immediately following a write SVC. If the flag is set when the file is created, the immediate write mode remains in effect until the file is deleted or until a utility operation resets the immediate write mode.

A user may specify blank compression when creating a sequential file. Blank compression saves disk space within a file by storing data in a more compact format.

Blank compression replaces strings of blanks by a count of blanks when writing to disk and restores the blank string when reading from disk. In operation, blank compression is not apparent to the user. It is generally advantageous to specify blank compression for files that usually contain many blanks, such as:

- Source files
- Listing files
- Text files

On the other hand, it is less advantageous (but not harmful) to use blank compression for files that contain few blanks, such as:

- Binary files (not ASCII data)
- Relocatable ASCII coded object files

A blank compressed record with no blanks requires one more word of disk space than if blank compression had not been specified.

Creating Sequential Files. A sequential file consists of logical records that are accessed in the sequence in which they appear in the file; that is, record 0 is accessed first, record 1 is accessed next, and so forth. Sequential files resemble files on magnetic tape. Examples of sequential file use include the following:

- As an input file of card images. If a logical record length of 80 is specified, the sequential file can function as a card reader to the program reading the file.
- As an output file, to function as the line printer.
- As a listing file for assembly or Link Editor listings.

The use of sequential files for assembly and Link Editor listings is a recommended practice. Assembly listings should be written to a sequential file. This saves time since the speed of the assembler is limited by the speed of the printer when the listing is printed directly. The use of sequential files for all input from and output to a device is usually preferable to direct input from or output to a device.

The logical length of records in a sequential file may be fixed or variable; variable length records may be any length including zero. When a file is created, the user may specify the logical record size or may allow the system to use the default size of 80 bytes. Specifying 0 causes the system to use the default of 80. If the user specifies the size, the size should be an estimate of the average logical record size of the records to be placed in the file.

The following characteristics apply to an example sequential file:

- 80-byte logical records
- 256-byte physical records

- 1,000-record initial allocation
- 500-record secondary allocation
- Blank compression

Although the logical record length is 80 bytes, few records contain that many bytes because blank compression reduces the length of each record by the number of blanks compressed from the record. The average number of bytes per record in this file is 40 bytes. (This average does not necessarily apply to any file other than this example.) Thus, since this is half the size of the actual logical record, the allocation sizes should be reduced by half.

The following is an example oı ıhe source code for the supervisor call block to create the example file and for the pathname of the file:

```
CRSEQ     DATA 0                          CREATE FILE PACK.USER.TEXT.FILE01,
          BYTE >90,0                      LRL = 80 AND PRL = 256. FILE IS A
          DATA 0,0                        BLANK COMPRESSED, EXPANDABLE,
          DATA 0,0                        SEQUENTIAL FILE. INITIAL ALLOCATION
          DATA 0,0                        IS 500, SECONDARY ALLOCATION IS 250.
          BYTE 0,>8D                      UTILITY FLAGS
          DATA 80
          DATA 256
          DATA PATHL
          DATA 0,0
          DATA 0,500
          DATA 0,250
PATHL     BYTE NAME-$-1                   PATHNAME LENGTH
          TEXT ‘PACK.USER.TEXT.FILE01’
NAME      EQU  $
```

The following example shows the same file created specifying a user ID and passcode:

```
CRSEQ     DATA  0                         CREATE FILE PACK.USER.TEXT.FILE01,
          BYTE  >90,0                     LRL = 80 AND PRL = 256. FILE IS A
          DATA  0,0                       BLANK COMPRESSED, EXPANDABLE,
          DATA  0,0                       SEQUENTIAL FILE. INITIAL ALLOCATION
          DATA  0,0                       IS 500, SECONDARY ALLOCATION IS 250.
          DATA  >018D                     UTILITY FLAGS WITH USER ID SPECIFIED
          DATA  80
          DATA  256
          DATA  PATHL
          DATA  USERID
          DATA  0
          DATA  0,500
          DATA  0,250
PATHL     BYTE  NAME-$-1                  PATHNAME LENGTH
          TEXT  ’PACK.USER.TEXT.FILE01’
NAME      EQU   $
```

```
USERID     BYTE  >13,0                  TOTAL LENGTH
           BYTE >02,>10                 SUBLIST TYPE AND LIST LENGTH
           TEXT 'IDFIELD'               USER ID
           TEXT '65420   '              USER PASSCODE
```

Creating Relative Record Files. A relative record file consists of records that can be accessed either randomly or sequentially. To access a record randomly, the user specifies a unique record number. For example, to access record number 5, the value 5 must be placed in the appropriate field of the I/O supervisor call block.

Relative record files can also be accessed sequentially. To access records sequentially, the user specifies a starting record number. File management automatically increments the record number after each read or write operation.

Relative record files are useful when each record in the file can be associated with a unique value ranging from 0 to any number. For example, in an inventory file, item number is appropriate for record number. In this case, to obtain information about item number 23456, access record number 23456.

The range of record numbers is one less than the number of records in the file. A relative record file can contain a maximum of 16,777,216 records. Because the records in a relative record file are fixed length, the system can convert a specified record number to a physical address on disk and can directly obtain any record with one disk access. Essentially, the location on the disk is derived from the calculation:

logical record position = file position + (record number × record length)

The logical record length for relative record files is fixed and is specified when the file is created. Specifying 0 causes the system to use the default size of 80 bytes.

Relative record files may be blocked or unblocked. Unless the immediate write mode applies, the physical record is not written to disk until the system needs the memory space occupied by the physical record or until the file is closed.

DNOS supports three special use categories of relative record files:

- Directory files
- Program files
- Image files

The following characteristics apply to an example relative record file:

- 72-byte logical records
- 256-byte physical records
- 1,000-record fixed size

The following is an example of the source code for the supervisor call block to create the example file and for the pathname of the file:

```
CRREL     DATA 0                    CREATE FILE VOL2.STO2.INV WITH
          BYTE >90,0                LRL = 72 AND PRL = 256. FILE IS
          DATA 0,0                  RELATIVE RECORD. FIXED SIZE IS
          DATA 0,0                  1000 RECORDS.
          DATA 0,0
          BYTE 0,>82                UTILITY FLAGS
          DATA 72
          DATA 256
          DATA PTHN
          DATA 0,0
          DATA 0,1000
          DATA 0,0
PTHN      BYTE NME-$-1              PATHNAME LENGTH
          TEXT 'VOL2.STO2.INV'
NME       EQU  $
```

Creating Key Indexed Files. Key indexed files are collections of records that are accessed by content, either sequentially or randomly. A primary key and optionally 1 through 13 secondary keys may be defined in each record. A key is a character string of defined length at a defined position in each record. Records of a key indexed file may be accessed in ascending or descending sequence of a specified key, or by specifying the content of a key.

Other characteristics of key indexed files are:

- A key may be defined as duplicatable; that is, more than one record may contain the same characters in the key.
- A key may be defined as modifiable; that is, a rewrite operation may alter the contents of the key.
- A key may consist of 1 through 100 characters and may overlap another key.
- A record may be accessed using only the first part of a key (partial key).
- A record may be read and locked pending update.
- The logical record length is variable and may change when a record is rewritten, except that the record length must be an even number greater than zero.
- Key indexed files are expandable and are blank compressed.
- A record to be modified is copied to a backup area, so that no data is lost in case of system failure. The operation being performed at the time of the failure is not applied to the file.

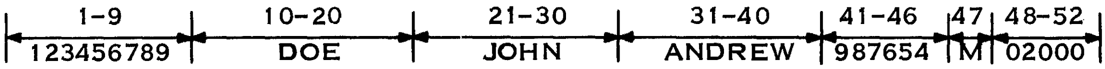
The keys for a key indexed file are defined when the file is created and apply to all records in the file. All of the data in the record may be included in one or more of the keys. Often, however, the keys are only a part of the entire record, and are used to access related data in the record. Keys may overlap other keys, be contiguous within the record, or be spread throughout the record.

A key indexed file is created in the immediate write mode regardless of the state of the write mode flag (bit 10 of the utility flags). Although the processing of the records of the file is significantly faster in the deferred write mode, the probability of loss of data when errors occur is much greater.

The Modify KIF Logging (MKL) command places a key indexed file in the deferred write mode. The file remains in the deferred write mode until a Close operation is performed to the file. Insert, Rewrite, and Delete operations are significantly faster in this mode because the number of disk read and write operations decreases. It is particularly appropriate to use the MKL command prior to copying a sequential file to a key indexed file using the Copy Sequential to Key (CSK) command or a user program that performs a similar function. Before processing a key indexed file in the deferred write mode, copy the key indexed file prior to entering the MKL command. You can rebuild the file if an error or system crash should occur while the file is in the deferred write mode using this copy. Perform the following steps if an error or system crash occurs:

1. Terminate all programs that modify the file.
2. Delete the key indexed file being processed in the deferred write mode.
3. Identify and correct the cause of the error or crash.
4. Using a copy of the original file, repeat the processing that failed. (If the file is in deferred write mode, retain *one good copy* of the original file in case another recovery is necessary.)

The following is an example of a key indexed file record. Each of the 52 characters in the record is included in one of the keys; several characters are included in two keys. Notice that a key contains similar information in each record.



2279583

Key	Columns	Definitions
1	41-46	Employee number
2	1-9	Social security number
3	10-20	Last name
4	21-30	First name
5	31-40	Middle name
6	10-40	Full name
7	47	Sex
8	48-52	Monthly salary

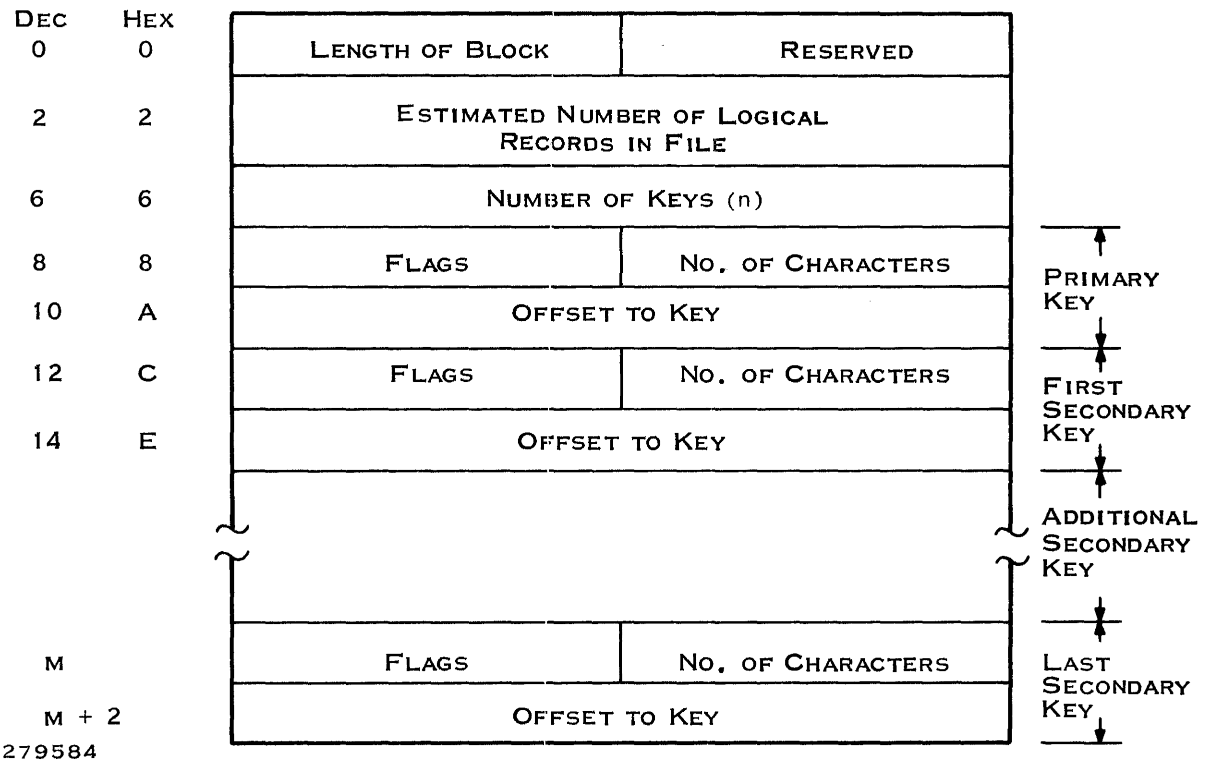
The employee number field is the primary key; seven secondary keys are defined. The records of the file can be accessed by any of the keys, in sequence. A record that contains a specific employee number, a specific social security number, or a specific full name can be accessed. An attempt to access a record that contains a specific monthly salary accesses the first record inserted into the file that contains that salary; successive accesses provide additional records of persons who receive that salary.

The employee number field is the primary key because it is the first key defined. The primary key may or may not be the first key in the record.

Each key may have two attributes; it may be duplicatable, and it may be modifiable. When a key is duplicatable, more than one record can contain the same data in the field. The name fields, the sex fields, and the monthly salary fields all should be duplicatable; it is logical to allow more than one record having the same value in these fields. The employee number and social security fields should not be duplicatable. It is not logical that more than one record would contain the same social security number or the same employee number. In a file that contains thousands of records with the same key value, operations that insert or delete and in some cases rewrite these records process much more slowly than if the duplicates did not exist.

The value of a key having the modifiable attribute may be changed by a rewrite operation. The last name field and the monthly salary field should be modifiable; these items may change. The employee number and social security number fields should not be modifiable; these values do not change. The primary key, by definition, is not modifiable. (A record that has incorrect data in a key that is not modifiable must be deleted and inserted again, with the correct data in the key.)

The key definition block defines the keys for a file. The address of this block is placed in bytes 12 and 13 of the supervisor call block for the Create File operation. The format of the key definition block is:



DEC	HEX			
0	0	LENGTH OF BLOCK	RESERVED	
2	2	ESTIMATED NUMBER OF LOGICAL RECORDS IN FILE		
6	6	NUMBER OF KEYS (n)		
8	8	FLAGS	NO. OF CHARACTERS	PRIMARY KEY
10	A	OFFSET TO KEY		
12	C	FLAGS	NO. OF CHARACTERS	FIRST SECONDARY KEY
14	E	OFFSET TO KEY		
				ADDITIONAL SECONDARY KEY
M		FLAGS	NO. OF CHARACTERS	LAST SECONDARY KEY
M + 2		OFFSET TO KEY		

2279584

The byte address of the data for the last key, m, is calculated as follows:

$$m = 4 \times (n - 1) + 8$$

The key definition block contains the following:

Byte	Contents
0	Length of the block, in bytes.
1	Reserved.
2–5	Estimated number of logical records. The file can expand beyond this number.
6–7	Number of keys for the file, 1 through 14.
	Bytes 8 through 11 define the primary key.

8 Flags for the key, as follows:
Bits 0-4 — Reserved.
Bit 5 — Zero for primary key. Modifiable flag for secondary keys.
Set as follows:
  1 — Key may be modified.
  0 — Key may not be modified.
Bit 6 — Reserved.
Bit 7 — Duplicatable flag. Set as follows:
  1 — Key may be duplicated.
  0 — Key is unique to one record.

9 Number of characters in key.

10-11 Offset to the key. Range is zero through number of characters in record.

Subsequent bytes, containing similar information, define the secondary keys, as applicable.

The size of a key indexed file may be calculated to assist in determining the disk space to be used by the file. The size of the file varies as records are inserted, rewritten, and deleted. Formulas and example calculations to estimate the size of a newly created key indexed file are shown in the *DNOS Systems Programmer's Guide.*

The following characteristics apply to an example key indexed file:

- 100-character average length of logical records
- 768-character physical records
- Two keys, neither key duplicatable or modifiable
- 10,000 logical records (estimated)

The following is an example of the source code for the supervisor call block to create the example file, for the key definition block, and for the pathname:

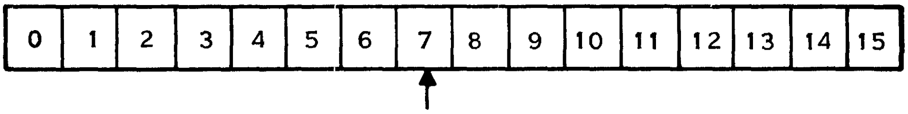
```
CRKEY   DATA 0              CREATE FILE VOL1.PAY WITH LRL =
        BYTE >90,0          100 AND PRL = 768. FILE IS KEY
        DATA 0,0            INDEXED. FILE IS EXPANDABLE WITH
        DATA 0,0            DEFAULT INITIAL AND SECONDARY
        DATA KEY,0          ALLOCATIONS.
        BYTE 0,>87          UTILITY FLAGS
        DATA 100
        DATA 768
        DATA PATH
        DATA 0,0
        DATA 0,0
        DATA 0,0
PATH    BYTE NAME-$-1       PATHNAME LENGTH
        TEXT 'VOL1.PAY'
NAME    EQU  $
```

```
KEY       BYTE  KEYEND-$-1          KEY BLOCK LENGTH
          DATA  0,10000             10000 MAXIMUM LOGICAL RECORDS
          DATA  2                   2 KEYS
          DATA  12                  1ST KEY 12 CHARACTERS
          DATA  0                   BEGINNING AT 1ST CHARACTER
          DATA  6                   2ND KEY 6 CHARACTERS
          DATA  5                   BEGINNING AT 6TH CHARACTER
KEYEND    EQU   $                   OVERLAPPING 1ST KEY
```

7.2.1.2 Deleting Files. To delete a file, a program executes an I/O Operations SVC with sub-opcode >92. The following fields of the utility supervisor call block apply:

- SVC opcode — >00
- Return code
- Utility sub-opcode — >92
- Utility flags
- Pathname address
- Parameter address

The following flags apply:



2285024

Bit 7 — parameter flag. Set as follows:
1 — Parameters are present and pointed to by bytes >18 and >19.
0 — No parameters

All other utility flags should be set to zero.

The parameter address may point to a user ID parameter as specified in the Performing Utility Functions paragraph in this section. If this option is specified in a secure environment, the file is deleted if the user has delete access to the file.

A Delete File operation does not delete the file if the file is either write protected or delete protected. A Remove Protection operation must be performed before a protected file may be deleted.

The pathname address is the address of an area of memory that contains the pathname of a file to be deleted. The first byte of the pathname area contains the number of characters in the pathname. Subsequent bytes contain the ASCII characters of the pathname.

The following is an example of the source code for a supervisor call block to delete the sequential file used as an example in the description of the Create File operation:

```
DFILE   DATA 0          DELETE FILE AT PATHNAME ADDRESS NAME.
        BYTE >92,0      PATHNAME BLOCK IS SHOWN IN CREATE
        DATA 0,0        FILE EXAMPLE.
        DATA 0,0
        DATA 0,0
        BYTE 0,0        UTILITY FLAGS
        DATA 0,0
        DATA NAME
        DATA 0,0
        DATA 0,0
        DATA 0,0
```

7.2.1.3 Assigning LUNOs. To assign a LUNO, a program executes an I/O Operations SVC with sub-opcode >91. The following fields of the utility supervisor call block apply:

- SVC opcode — >00
- Return code
- Utility sub-opcode — >91
- Logical unit number (LUNO)
- <Resource type>
- Utility flags
- Pathname address
- Parameter address

An Assign LUNO operation may request the autocreate option, which first creates the file, then assigns a LUNO to it. All fields and utility flags defined for a Create File operation also apply to an Assign LUNO with autocreate operation.

The system returns the resource type in bytes 6 and 7 of the call block. The resource type is one of the following hexadecimal numbers:

Type	**Resource**
0101	Sequential file
0201	Relative record file
0301	Key indexed file
0401	Directory file
0501	Program file
0601	Image file

The following utility flags apply:

0	1–2	3–4	5	6	7	8	9	10	11–12	13	14–15

2279585

Bit 0 — <Created by assign>. Set to one by system after creating the file (autocreate option).

Bits 1-2 — File usage flag. Set as follows:
00 — No special usage.
01 — Directory file.
10 — Program file.
11 — Image file.

Bits 3-4 — Scope of LUNO flag. Set as follows:
00 — Task-local LUNO.
01 — Job-local LUNO.
10 — Global LUNO.
11 — Job-local-shared LUNO.

Bit 5 — Generate LUNO flag. Set as follows:
1 — Assign the next available LUNO and return LUNO in byte 3.
0 — Assign the LUNO specified in byte 3.

Bit 6 — Autocreate flag. Set as follows:
1 —Create the file, if it does not already exist (call block must specify file parameters).
0 —Do not create file.

Bit 7 — Parameter flag. Set as follows:
1 — Parameters are present and pointed to by bytes >18 and >19.
0 — No parameters

Bit 8 — LRL flag (autocreate option). Set as follows:
1 — Place the logical record length in bytes 12 and 13.
0 —Place the logical record length in bytes >8 and >9. It is recommended that you set the LRL flag to one and place the logical record length in bytes >12 >13.

Bit 9 — Temporary file flag (autocreate option). Set as follows:
1 — Temporary file. When pathname address in bytes 22 and 23 is zero, the temporary file is placed on the system disk.
0 — Not a temporary file.

Bit 10 — Write mode flag (autocreate option). Set as follows:
1 — Immediate write mode.
0 — Deferred write mode (normal mode).
The setting of the write mode flag does not apply to a key indexed file, which is always created in the immediate write mode.

Bits 11-12 — Data format flag (autocreate option). Set as follows:

- 00 — Normal record image.
- 01 — Blank compressed.

  The setting of the data format flag does not apply to a key indexed file, which is always blank compressed.

Bit 13 — Allocation flag (autocreate option). Set as follows:

- 1 — Expandable file.
- 0 — Fixed size file.

  If the file already exists, the system returns this value.

Bits 14-15 — File type flag (autocreate option). Set as follows:

- 01 — Sequential file.
- 10 — Relative record file.
- 11 — Key indexed file.

  If the file already exists, the system returns this value.

A logical unit number (LUNO) must be assigned to an I/O resource to identify the resource for an I/O operation. The scope of a global LUNO is not limited to a single job or task. The LUNO applies in all jobs and tasks executing while it remains assigned. The scope of a job-local LUNO is limited to the tasks in the job. A job-local LUNO is assigned by one of the tasks in the job or by an SCI command. The scope of a task-local LUNO is limited to the task that assigns the LUNO. A task-local LUNO is assigned by a task.

Job-local-shared LUNOs (shared LUNOs) are job-local LUNOs that can be used by more than one task within a given job. Each task that uses the LUNO must open it. The access privileges of the LUNO are compared to those requested in the Open operation. If the Open operation requests greater access privileges and it does not conflict with the access privileges of other LUNOs that are assigned and opened to the resource, the privilege level of the LUNO is changed to the greater value. The access privileges of a LUNO in order of increasing value are read only, shared, exclusive write, exclusive all. If the requested access privilege is less than or equal to the present value, the privilege level of the LUNO is not changed. Thus, all tasks that use a shared LUNO have the same access privileges to the resource regardless of how they opened it.

A count of the number of successful Open operations is kept. The same number of Close operations must be performed before the LUNO can be released. If a Close operation is not performed, the LUNO is not released until the job terminates.

The use of shared LUNOs tends to reduce the total number of LUNOs required in the system. This type of LUNO is not recommended for sequential files because there is no defined method of positioning the file; that is, the task has no control of which record is read or written.

When assigning a LUNO to a directory, program file, or image file, the file usage flags must be set. This prevents access to the file that is not compatible with the defined use of the file.

If a user ID parameter is specified in a secure environment as described in the Performing Utilities Functions paragraph in this section, the LUNO is assigned only if the user ID in the parameter list has rights to the file. Subsequent I/O operations using the LUNO are verified against the specified user's access rights.

The Assign LUNO operation may assign the next available LUNO or a LUNO specified in the LUNO field. When the generate LUNO flag is set to one, the system assigns the next available LUNO and returns the number in the LUNO field. When the flag is set to zero, the system considers the contents of the LUNO field of the supervisor call block to be the desired LUNO.

The autocreate option combines creating the file with assigning a LUNO to the file. When the autocreate flag is set to one and the file does not already exist, the system creates a file, using the contents of the supervisor call block. The flags and fields defined for a create file operation must be set to valid values. The system sets the created-by-assign flag to one when it has successfully created the file, and assigns the LUNO.

The autocreate option creates a task level temporary file when the temporary file bit is set. The pathname of the temporary file is automatically asssigned by the system. A pathname, containing the name of the disk that will contain the temporary file, can be specified. The default is the system disk. The system deletes the file when the LUNO is released.

The pathname address is the address of an area of memory that contains the pathname of the file to which the LUNO is assigned. The first byte of the pathname area contains the number of characters in the pathname. Subsequent bytes contain the ASCII characters of the pathname.

The following is an example of the source code for a supervisor call block and the pathname block to assign a LUNO to a file:

```
ALUNO   DATA 0                    ASSIGN TASK LOCAL LUNO >18 TO
        BYTE >91                  FILE VOL3.BILL.INPUT
        BYTE >18
        DATA 0,0
        DATA 0,0
        DATA 0,0
        BYTE 0,0                  UTILITY FLAGS
        DATA 0,0
        DATA PNME
        DATA 0,0
        DATA 0,0
        DATA 0,0
PNME    BYTE N4-$-1               PATHNAME LENGTH
        TEXT 'VOL3.BILL.INPUT'
N4      EQU  $
```

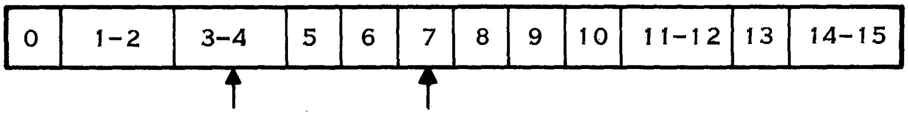
The following is an example of the source code for a supervisor call block and a pathname block to autocreate and assign a LUNO to a file:

```
ALCR      DATA 0                  ASSIGN JOB LOCAL LUNO >25 TO
          BYTE >91                FILE VOL6.USER.REP; AUTOCREATE
          BYTE >25                THE FILE. FILE IS
          DATA 0,0                SEQUENTIAL, FIXED LENGTH,
          DATA 0,0                BLANK COMPRESSED. LRL = 50,
          DATA 0,0                PRL = 256.
          BYTE >0A,>89            UTILITY FLAGS
          DATA 50
          DATA 256
          DATA FNME
          DATA 0,0
          DATA 0,0
          DATA 0,0
FNME      BYTE FN-$-1             PATHNAME LENGTH
          TEXT 'VOL6.USER.REP'
FNME      EQU  $
```

7.2.1.4 Releasing LUNOs. To release a LUNO, a program executes an I/O Operations SVC with sub-opcode >93. The following fields of the utility supervisor call block apply:

- SVC code — 0
- Return code
- Utility sub-opcode — >93
- Logical unit number (LUNO)
- Utility flags

The following utility flags apply:



2279586

Bits 3-4 — Scope of LUNO. Set as follows:
00 — Task-local LUNO.
01 — Job-local LUNO.
10 — Global LUNO.
11 — Job-local-shared LUNO.

Bit 7 — Parameter flag. Set as follows:
1 — Parameters are present and pointed to by bytes >18 and >19.
0 — No parameters.

Set all other utility flags to zero.

A Release LUNO operation does not release a LUNO that has a different scope from that specified by the scope of LUNO flag. For example, if global LUNO >23, job-local LUNO >23, and task-local LUNO >23 were all assigned, and a Release LUNO operation for task-local LUNO >23 were performed, the global and job-local LUNOs would remain assigned.

In a networking environment, an Assign LUNO-Release LUNO pair bounds a session. If a user desires a session to span more than one such stage, it is possible to designate a release LUNO parameter that tells the local area network that this release LUNO does not terminate the session. The parameter must be included in the parameter list pointed to by the parameter address field in the call block. The parameter flag bit must be set in the utility flags. A parameter block which includes the release LUNO parameter only is shown below:

DEC	HEX		
0	0	>04	>00
2	2	>05	>01
4	4	>03	

2285025

Byte	Contents
0	Length of the parameter list minus the length byte in bytes
1	Reserved — Must be set to zero
2	>05 — Release LUNO parameter sublist number
3	Length of the Release LUNO sublist in bytes
4	Parameter value: >03 — Parameter enabled >02 — Parameter disabled

The following is an example of the source code for a supervisor call block to release a LUNO:

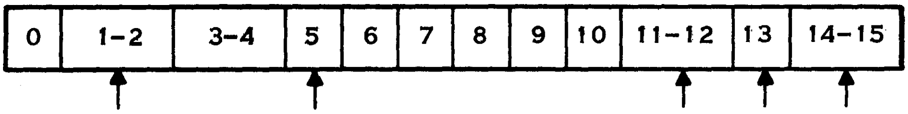
```
RLUNO   DATA 0              RELEASE GLOBAL LUNO >23.
        BYTE >93
        BYTE >23
        DATA 0,0
        DATA 0,0
        DATA 0,0
```

```
BYTE >10,0                 UTILITY FLAGS
DATA 0,0
DATA 0
DATA 0,0
DATA 0,0
DATA 0,0
```

7.2.1.5 Verifying Pathnames. To verify a pathname, a program executes an I/O Operations SVC with sub-opcode >99. The following fields of the utility supervisor call block apply:

- SVC code — 0
- Return code
- Utility sub-opcode — >99
- Data buffer address
- Utility flags
- Pathname address

The following utility flags apply:



2279587

Bits 1-2 — <File usage flag>. Set by DNOS to the usage of the verified file as follows:
00 — No special usage.
01 — Directory file.
10 — Program file.
11 — Image file.

Bit 5 — <System flag>. Set by DNOS to 1.

Bits 11-12 — <Data format flag>. Set by DNOS to the format of the verified file as follows:
00 — Normal record image.
01 — Blank compressed.

Bit 13 — <Allocation flag>. Set by DNOS to the allocation of the verified file as follows:
1 — Expandable file.
0 — Fixed size file.

Bits 14-15 — <File type flag>. Set by DNOS to the type of the verified file as follows:
01 — Sequential file.
10 — Relative record file.
11 — Key indexed file.

Set all other flags to zero.

The Verify Pathname operation performs a syntax check on the pathname. When the pathname is that of an existing file, the system returns the file usage, file type, data format, and allocation type in the corresponding flags of the utility flag word. The file type is returned in the data buffer address field in the form returned for an Assign LUNO operation. When the pathname is of a nonexistent file, the appropriate error code is returned and the utility flag word is cleared.

The pathname address is the address of an area of memory that contains the pathname to be verified. The byte at the pathname address contains the number of characters in the pathname. Subsequent bytes contain the ASCII characters of the pathname.

The system returns the resource type in bytes 6 and 7 of the call block. The resource type is one of the following hexadecimal numbers:

Type	**Resource**
0101	Sequential file
0201	Relative record file
0301	Key indexed file
0401	Directory file
0501	Program file
0601	Image file

The following is an example of the source code for a supervisor call block to verify a pathname:

```
VFY     DATA 0                    VERIFY PATHNAME VOL1.USER.SOURCE
        BYTE >99,0
        DATA 0,0
        DATA 0,0
        DATA 0,0
TYPE    BYTE 0,0                  UTILITY FLAGS
        DATA 0,0
        DATA NA
        DATA 0,0
        DATA 0,0
        DATA 0,0
NA      BYTE NA1-$-1              PATHNAME LENGTH
        TEXT 'VOL1.USER.SOURCE'
NA1     EQU  $
```

7.2.1.6 Renaming Files. To assign a new pathname to a file, a program executes an I/O Operations SVC with sub-opcode >95. The following fields of the utility supervisor call block apply:

- SVC code — 0
- Return code
- Utility sub-opcode — >95
- Logical unit number (LUNO)

- User flags
- Utility flags
- Pathname address

All utility flags should be set to zero.

The Do Not Replace flag (bit 5) in the user flag field, applies to the Rename operation.

0	1	2	3	4	5	6	7
					↑		

2279588

Bit 5 — Do Not Replace flag. Set as follows:
1 — When a file already exists under the new pathname, do not rename the file.
0 — Rename the file, replacing an existing file if one exists.

The following utility flags apply to the rename operation:

0	1–2	3–4	5	6	7	8	9	10	11–12	13	14–15
					↑						

2283208

Bit 7 — Parameter flag. Set as follows:
1 — Parameters are present and pointed to by bytes >18 and >19.
0 — No parameters

Set all other utility flags to zero.

The operation assigns a new pathname to the file assigned to the LUNO specified in the LUNO field. When a file having the new pathname already exists, and the do not replace flag is set to zero, the existing file is deleted but any aliases of the new pathname remain valid for the new pathname. When the do not replace flag is set to one, a file having the new pathname is not deleted, and the pathname of the file assigned to the specified LUNO is not changed.

A Rename operation can change a filename. This requires changing the name only in the appropriate directory. If the directory is full, however, the Rename operation fails and returns an error, meaning that it cannot successfully create a file. A Rename operation can change the name of any directory in the pathname, except that the Rename operation may not replace a directory that contains one or more files or directories. Changing a directory name requires changes in the entries in one or more directories. A Rename operation cannot change the volume name, which would require copying the file and possibly the directories to another volume.

If the file security feature was specified during system generation, a Rename File SVC of a directory is not allowed unless specified by a member of the system manager access group.

Some special conditions exist when the Rename operation is used to change the name of a program file in a directory. In a Rename operation the names of channels assigned to the source program file are also copied and are associated with the new destination program file. Aliases of the source program file are not copied. Any aliases assigned to the renamed program filename are associated with the new destination program filename.

The user cannot specify the scope of the LUNO for the Rename operation. The routine that performs the operation searches the LUNO list, which contains the task-local LUNOs, then the job-local LUNOs, followed by the global LUNOs. To rename a file assigned to a job-local LUNO, the task-local LUNO of the same value cannot be assigned. To rename a file assigned to a global LUNO, neither the task-local LUNO nor the job-local LUNO of the same value can be assigned. That is, when a Rename operation specifies LUNO >35, the operation is attempted for any resource assigned to task-local LUNO >35. If task-local LUNO >35 is not currently assigned, the operation is attempted for job-local LUNO >35. If neither task-local LUNO >35 nor job-local LUNO >35 is currently assigned, the operation is attempted for global LUNO >35.

Any LUNO assigned to the file to be renamed (other than the one specified in the LUNO field) must be released prior to renaming the file. When the Rename operation replaces an existing file, any LUNO assigned to the existing file must be released prior to the Rename operation.

When the Rename File SVC executes, the resulting file retains the security access rights of the original file. For example, when a file named LIST1 is renamed to LISTS2, LIST2 retains all the security access rights of LIST1.

There is a Rename File SVC option that assigns the security access rights of the destination file to the input file. That is, if files LIST1 and LIST2 have different security access rights and LIST1 is renamed to LIST2, the resulting file retains the security access rights of the original LIST2. This option is available only on systems where the security feature was chosen during system generation.

This option operates only when the following conditons are true:

- The destination file already exists.

- The replace option is specified.

If the destination file does not exist, the resulting file retains the security access rights of the original file.

To use this option, it must be specified as an entry in a parameter list and the parameter flag must be set. The parameter address field must point to the parameter block. The following diagram shows the parameter list block:

DEC	HEX		
0	0	>04	>00
2	2	>04	>01
4	4	>03	

2285267

Byte	Contents
0	Total parameter list length minus the length byte in bytes
1	Reserved — must be set to zero
2	>04 — Rename File parameter type
3	Length of the Rename File sublist in bytes
4	Sublist value: >03 — Parameter enabled >02 — Parameter disabled

The pathname address is the address of an area of memory that contains the new pathname. The byte at the pathname address contains the number of characters in the pathname. Subsequent bytes contain the ASCII characters of the pathname.

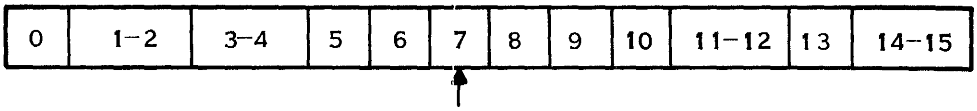
The following is an example of the source code for a supervisor call block to assign a new pathname to a file using the Rename File parameter:

```
RNAME   DATA 0                      RENAME FILE ASSIGNED TO
        BYTE >95                    LUNO >25. DO NOT REPLACE.
        BYTE >25                    NEW NAME IS VOL4.USER2.SOURCE
        DATA 0,0                    KEEP SECURITY OF NEW NAME
        DATA 0,0
        DATA 0,0
        BYTE 0,>04                  UTILITY FLAGS
        DATA 0,0
        DATA NNAM
        DATA RENAME
        DATA 0,0
        DATA 0,0,0
NNAM    BYTE NN1-$-1                PATHNAME LENGTH
        TEXT 'VOL4.USER2.SOURCE'
NN1     EQU  $
RENAME  BYTE >04,0                  TOTAL LENGTH
        BYTE >04,>01
        BYTE >03
```

7.2.1.7 Write Protecting Files. To write protect a file, a program executes an I/O Operations SVC with sub-opcode >97. The following fields of the utility supervisor call block apply:

- SVC opcode — >00
- Return code
- Utility sub-opcode — >97
- Utility flags
- Pathname address
- Parameter address

The following utility flags apply:



2283208

Bit 7 — Parameter flag. Set as follows:
1 — Parameters are present and pointed to by bytes >18 and >19.
0 — No parameters

Set all other utility flags to zero.

Files are created with no protection. After a Write Protect operation is performed, neither a Write nor a Delete operation may be performed on the file. Protection is removed by performing a Remove Protection operation.

In a secure environment, a user ID can be specified as described in the File Utility Operations paragraph in this section. If the user ID in the parameter list has delete and write access, this SVC write protects the file.

The pathname address is the address of an area of memory that contains the pathname of the file to be write-protected. The byte at the pathname address contains the number of characters in the pathname. Subsequent bytes contain the ASCII characters of the pathname.

The following is an example of the source code for a supervisor call block to write protect a file:

```
WRPR    DATA 0                      APPLY WRITE AND DELETE
        BYTE >97                    PROTECTION TO FILE
        BYTE 0                      VOL5.SOURCE.PROGA.
        DATA 0
        DATA 0
        DATA 0,0
        DATA 0,0
        BYTE 0,0                    UTILITY FLAGS
        DATA 0,0
        DATA PNAM
        DATA 0,0
        DATA 0,0
        DATA 0,0
PNAM    BYTE PN1-$-1                PATHNAME LENGTH
        TEXT 'VOL5.SOURCE.PROGA'
PN1     EQU  $
```

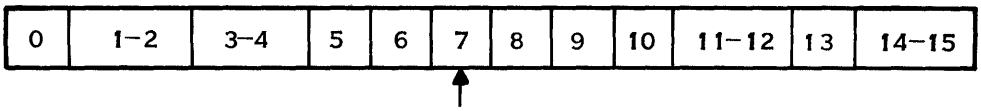
7.2.1.8 Delete Protecting Files. To delete protect a file, a program executes an I/O Operations SVC with sub-opcode >98. The following fields of the utility supervisor call block apply:

- SVC opcode — >00

- Return code

- Utility sub-opcode — >98
- Utility flags
- Pathname address
- Parameter address

The following utility flags apply:



2283208

Bit 7 — Parameter flag. Set as follows:
1 — Parameters are present and pointed to by bytes >18 and >19.
0 — No parameters

Set all other utility flags to zero.

Files are created with no protection. After a Delete Protect operation is performed, a Delete operation may not be performed on the file. Protection is removed by performing a Remove Protection operation.

The pathname address is the address of an area of memory that contains the pathname of the file to be delete protected. The byte at the pathname address contains the number of characters in the pathname. Subsequent bytes contain the ASCII characters of the pathname.

In a secure environment, a user ID can be specified as described in the File Utility Operations paragraph in this section. This SVC delete protects the file if the user ID has write and delete access to the file.

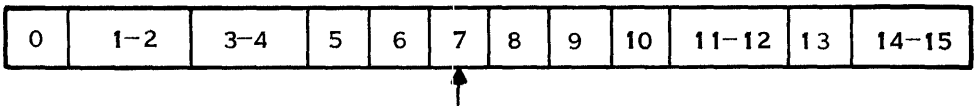
The following is an example of the source code for a supervisor call block to delete protect a file:

```
DPR      DATA 0                       APPLY DELETE PROTECTION
         BYTE >98                     TO FILE VOL1.SOURCE.PROGB
         BYTE 0
         DATA 0
         DATA 0
         DATA 0,0
         DATA 0,0
         BYTE 0,0                     UTILITY FLAGS
         DATA 0,0
         DATA DPNAM
         DATA 0,0
         DATA 0,0
         DATA 0,0
DPNAM    BYTE DPN1-$-1                PATHNAME LENGTH
         TEXT 'VOL1.SOURCE.PROGB'
DPN1     EQU  $
```

7.2.1.9 Removing File Protection. To remove protection from a file, a program executes an I/O Operations SVC with sub-opcode >96. The following fields of the utility supervisor call block apply:

- SVC opcode — >00
- Return code
- Utility sub-opcode — >96
- Utility flags
- Pathname address
- Parameter address

The following utility flags apply:



2283208

Bit 7 — Parameter flag. Set as follows:
1 — Parameters are present and pointed to by bytes >18 and >19.
0 — No parameters

Set all other utility flags to zero.

When write or delete protection is applied to a file, the file remains protected until a Remove Protection operation is performed on the file. The Remove Protection operation removes both write and delete protection, leaving the file unprotected.

The pathname address is the address of an area of memory that contains the pathname of the file from which protection is to be removed. The byte at the pathname address contains the number of characters in the pathname. Subsequent bytes contain the ASCII characters of the pathname.

In a secure environment, a user ID can be specified as described in the File Utility Operations paragraph in this section. This SVC removes the file delete and write-protection if the user ID has write and delete access to the file.

The following is an example of the source code for a supervisor call block to remove protection from a file:

```
RPROT   DATA 0              REMOVE PROTECTION FROM FILE USING
        BYTE >96            PATHNAME BLOCK OF PRECEDING
        BYTE 0              EXAMPLE
        DATA 0
        DATA 0
        DATA 0,0
        DATA 0,0
        BYTE 0,0            UTILITY FLAGS
        DATA 0,0
        DATA DPNAM
        DATA 0,0
        DATA 0,0
        DATA 0,0
```

7.2.1.10 Adding an Alias. To add an alias for a file, a program executes an I/O Operations SVC with sub-opcode >9A. The following fields of the utility supervisor call block apply:

- SVC code — 0
- Return code
- Utility sub-opcode — >9A
- Logical unit number (LUNO)
- Pathname address

All utility flags should be set to zero.

An alias is an alternate filename or directory name that may be assigned to allow access to the file by an alternate pathname. An alias may be assigned at any directory level other than the volume directory level. The original pathname of the file remains valid.

The pathname address is the address of an area of memory that contains the alias pathname. The first byte contains the number of characters in the alias pathname. Subsequent bytes contain the characters of the pathname. The alias pathname consists of the volume name of the pathname, followed by directory names as required, followed by the alias being added. Elements of the alias pathname are separated by periods. The alias pathname must contain any directories of the pathname that precede the alias.

The logical unit number is the LUNO assigned to the pathname for which an alias is to be added. The pathname must have the same number of elements as the alias pathname; it must end with the directory name or filename for which the alias is being added.

A pathname that ends with a directory name is a directory pathname. Set the file usage flag in the utility flags word to 01 (directory file) when you assign a LUNO to the pathname.

For example, the pathname of a file is:

VOL2.PROJA.HICKS.SOURCE.PROGA

To add alias FORMAT for the filename PROGA, assign a LUNO to the file pathname and add an alias using alias pathname:

VOL2.PROJA.HICKS.SOURCE.FORMAT

To add alias COML for directory HICKS, assign a LUNO to directory file pathname:

VOL2.PROJA.HICKS

Then add the alias using alias pathname:

VOL2.PROJA.COML

The following is an example of the source code for a supervisor call block to add an alias:

```
AALIAS   DATA 0                   ADD AN ALIAS FOR FILE ASSIGNED TO
         BYTE >9A                 LUNO >47.
         BYTE >47
         DATA 0
         DATA 0
         DATA 0,0
         DATA 0,0
         BYTE 0,0                 UTILITY FLAGS
         DATA 0,0
         DATA ALPNM
         DATA 0,0
         DATA 0,0
         DATA 0,0
ALPNM    BYTE ALP-$-1             PATHNAME LENGTH
         TEXT 'VOL1.IN'           ALIAS
ALP      EQU  $
```

7.2.1.11 Deleting an Alias. To delete an alias for a file, a program executes an I/O Operations SVC with sub-opcode >9B. The following fields of the utility supervisor call block apply:

- SVC code — 0
- Return code
- Utility sub-opcode — >9B
- Pathname address

All utility flags should be set to zero.

An alias is an alternate filename or directory name that allows access to the file by an alternate pathname. A Delete Alias operation removes the specified alias from the directory.

The pathname address is the address of an area of memory that contains the alias pathname. The alias pathname is identical to the alias pathname used to add the alias.

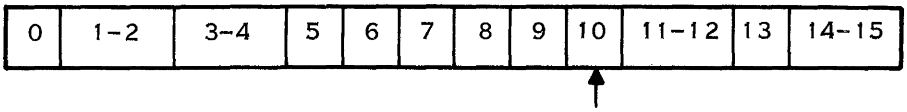
The following is an example of the source code for a supervisor call block to delete an alias:

```
DALIAS    DATA 0                        DELETE THE ALIAS ADDED IN THE
          BYTE >9B                      PRECEDING EXAMPLE
          BYTE 0
          DATA 0
          DATA 0
          DATA 0,0
          DATA 0,0
          BYTE 0,0                      UTILITY FLAGS
          DATA 0,0
          DATA ALPNM
          DATA 0,0
          DATA 0,0
          DATA 0,0
```

7.2.1.12 Specifying the Write Mode. To specify the write mode for a file, a program executes an I/O Operations SVC with sub-opcode >9C. The following fields of the utility supervisor call block apply:

- SVC code — 0
- Return code
- Utility sub-opcode — >9C
- Logical unit number (LUNO)

The following utility flag applies:



2279589

Bit 10 — Write mode flag. Set as follows:
0 — Deferred write mode.
1 — Immediate write mode.

Set all other utility flags to zero.

The write mode is either the immediate or the deferred write mode. The deferred write mode is the normal mode, because it is more efficient; writing does not actually occur until the system requires the memory space occupied by the buffer that contains the physical record to be written. When the immediate write mode is specified, the physical record is written each time a logical record within the physical record is written. The immediate write mode provides more certain and more accurate error detection and identification and reduces the amount of data lost in the event of system failure.

The write mode of the file assigned to the LUNO entered in the supervisor call block is specified by the operation. When the same LUNO is assigned at more than one level of scope, the LUNO applies in this order: task-local LUNO, job-local LUNO, and global LUNO. The write mode flag is set to one for the immediate write mode, or to zero for the deferred write mode.

Using the immediate write mode updates the file and its file structure each time a write operation is performed. This mode maintains file integrity, but it costs more in I/O execution time than the deferred write mode.

The following is an example of the source code for a supervisor call block to set the write mode of a file:

```
SWMODE  DATA 0              SET THE WRITE MODE FLAG TO IMMEDIATE
        BYTE >9C            FOR THE FILE ASSIGNED TO LUNO >23
        BYTE >23
        DATA 0,0
        DATA 0,0
        DATA 0,0
        BYTE 0,>20          UTILITY FLAGS
        DATA 0,0
        DATA 0
        DATA 0,0
        DATA 0,0
        DATA 0,0
```

7.3 SEQUENTIAL FILE I/O

Sequential file I/O uses the following basic supervisor call block to effect I/O transfers, file positioning, and other I/O operations.

SVC >00 -- I/O Operations

Align on word boundary
Can be initiated as an event

Dec	Hex		
0	0	>00	<Return Code>
2	2	Sub-Opcode	LUNO
4	4	<System Flags>	User Flags
6	6	Data Buffer Address	
8	8	Read Character Count	
10	A	Write Character Count/ < Actual Read Count >	

2279470

The following sub-opcodes apply to sequential files:

00	Open
01	Close
02	Close, Write EOF
03	Open and Rewind
04	Close and Unload
05	Read File Characteristics
06	Forward Space
07	Backward Space
09	Read ASCII
0B	Write ASCII
0D	Write EOF
0E	Rewind
10	Rewrite
11	Modify Access Privileges
12	Open Extend
4A	Unlock Record
59	Multiple Record Read
5B	Multiple Record Write

The following sub-opcodes perform operations identical to those shown:

Sub-opcode	**Operation**	**Identical to**
0A	Read Direct	Read ASCII
0C	Write Direct	Write ASCII

The following sub-opcodes are ignored:

08	Not used
0F	Unload

Except for the Read File Characteristics operation, the file must be opened using sub-opcode >00, >03, or >12 prior to any I/O operation.

7.3.1 Open

Sub-opcode >00 specifies an Open operation. The Open operation enables the calling task to perform I/O operations on the file assigned to the specified LUNO. If the Open operation is successful, the access privilege requested in bits 3 and 4 of byte 5 is granted to the task. An Open operation must be performed before a task can perform any I/O operation other than Read File Characteristics.

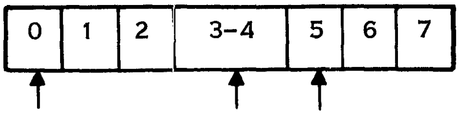
An Open operation does not alter the file position (the next record to be accessed). An Assign LUNO operation positions the file at the first record. A Close operation leaves the file positioned as it was positioned following the most recent access to the file.

The following fields of the basic call block apply to an Open operation:

- SVC code — 0

- Return code
- Sub-opcode — >00
- Logical unit number (LUNO)
- User flags
- Data buffer address
- Read character count

The following user flags apply to an Open operation:



2279590

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

Bits 3-4 — Access privilege flag. Set as follows:
00 — Exclusive write.
01 — Exclusive all.
10 — Shared.
11 — Read only.

Bit 5 — Do not replace flag. Applies to shared, exclusive write, and exclusive all access only. Set as follows:
1 — Open file that was created with the autocreate option when the LUNO was assigned. When file existed prior to the Assign LUNO operation, terminate in error.
0 — Open file regardless of when it was created.

The LUNO field contains the LUNO assigned to the file to be opened.

The Open operation returns the file type code in the buffer address field. The file type code for a sequential file is >01FF.

When the calling task places zero in the read character count field, the Open operation returns the logical record length specified for the file.

The following is an example of the source code for a supervisor call block to open a sequential file:

```
OSF     DATA 0             OPEN FILE ASSIGNED TO LUNO >4C
        BYTE 0,>4C         WITH SHARED ACCESS
        BYTE 0,>10
SFT     DATA 0
BL      DATA 0
        DATA 0
```

7.3.2 Close

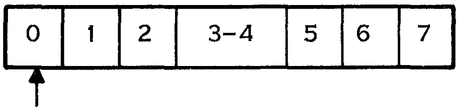
Sub-opcode >01 specifies a Close operation. The Close operation ends I/O to a LUNO from the calling task. The LUNO remains assigned to the file. Specifically, for the file assigned to the LUNO, the Close operation:

- Unlocks any locked records
- Writes all modified file blocks on which write was deferred
- Releases access privileges
- Updates file data structures maintained by the system to accurately describe the current file. Until the Close operation is performed, data structures on disk do not accurately reflect the contents of the file. If a system crash occurs before a Close is performed, new records written to the end of an existing file will be lost.

The following fields of the basic supervisor call block apply to a Close operation:

- SVC code — 0
- Return code
- Sub-opcode — >01
- Logical unit number (LUNO)
- User flags

The following user flag applies to a Close operation:



2279591

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

The LUNO field contains the LUNO assigned to the file to be closed.

The following is an example of the source code for a supervisor call block to close a file:

```
CSF      DATA 0                   CLOSE FILE ASSIGNED TO LUNO >4C
         BYTE >01,>4C
         BYTE 0,>10
         DATA 0
         DATA 0
         DATA 0
```

7.3.3 Close, Write EOF

Sub-opcode >02 specifies a Close, Write EOF operation. A Close, Write EOF operation consists of a Write EOF operation followed by a Close operation.

7.3.4 Open and Rewind

Sub-opcode >03 specifies an Open and Rewind operation. An Open and Rewind operation performs an Open operation followed by a Rewind operation.

7.3.5 Close and Unload

Sub-opcode >04 specifies a Close and Unload operation. The Close and Unload operation is the same as a Close operation.

7.3.6 Read File Characteristics

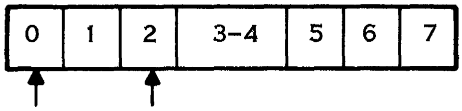
Sub-opcode >05 specifies a Read File Characteristics operation. The Read File Characteristics operation returns file characteristics information in a buffer specified by the user. The file characteristics consist of 10 bytes of information.

In a secure environment, this operation can be used to determine the access rights that a user has to a file. If this option is used, a word of file-access rights is returned.

The following fields of the basic supervisor call block apply to a Read File Characteristics operation:

- SVC code — 0
- Return code
- Sub-opcode — >05
- Logical unit number (LUNO)
- User flags
- Data buffer address
- Read character count
- <Actual read count>

The following user flags apply to a Read File Characteristics operation:



2279592

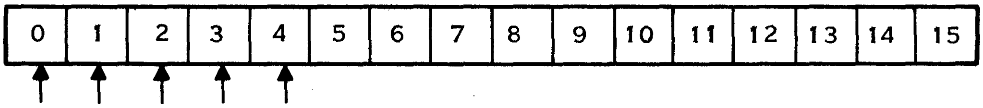
Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.
Bit 2 — Security access rights flag. Set as follows:
1 — System returns a word of rights in the buffer specified by the user.
0 — System returns file characteristics.

The LUNO field contains the LUNO assigned to the file for which characteristics are to be read.

The data buffer address is the address of the buffer into which DNOS places the file characteristics. The buffer should contain at least 10 bytes if the security rights option is not used. If the security rights option is used, the buffer should contain at least two bytes.

The read character count is the length of the buffer.

If the security access rights flag is set, a word of file access rights is returned to the buffer specified by the user. The following explains the meaning of the bits in the returned word:



2285027

Bit 0 — 1 if the user has read access
Bit 1 — 1 if the user has write access
Bit 2 — 1 if the user has delete access
Bit 3 — 1 if the user has execute access
Bit 4 — 1 if the user has control access

If the security access rights flag is not set, 10 bytes of file characteristics are returned.

DNOS returns the number of characters stored in the buffer in the actual read count field. The file characteristics for a sequential file consist of 10 characters. The contents of the buffer following a Read File Characteristics operation are:

DEC	HEX	
0	0	FILE ATTRIBUTE FLAGS
2	2	PHYSICAL RECORD LENGTH
4	4	LOGICAL RECORD LENGTH
6	6	NUMBER OF LOGICAL RECORDS
8	8	

2279593

Byte **Contents**

0-1 File attribute flags, as follows:
Bits 0-1 — File usage:
00 — No special usage.

Bits 2-3 — Data format:
00 — Not blank compressed.
01 — Blank compressed.

Bit 4 — Allocation type:
0 — Fixed size file.
1 — Expandable file.

Bits 5-6 — File type:
01 — Sequential.

Bit 7 — Write protection flag:
0 — Not write protected.
1 — Write protected.

Bit 8 — Delete protection flag:
0 — Not delete protected.
1 — Delete protected.

Bit 9 — Temporary file flag:
0 — Permanent file.
1 — Temporary file.

Bit 10 — Blocked file flag:
0 — Blocked.
1 — Not blocked.

Bit 11 — Reserved.

Byte	Contents
	Bit 12 — Write mode flag: 0 — Deferred write. 1 — Immediate write.
	Bits 13–15 — Reserved.
2–3	Physical record length.
4–5	Logical record length.
6–9	Number of logical records.

The following is an example of the source code for a supervisor call block to read file characteristics, and for the required buffer:

```
RSFC     DATA 0                       READ CHARACTERISTICS OF FILE
         BYTE >05,>4C                 ASSIGNED TO LUNO >4C
         BYTE 0,0
         DATA SFC
         DATA 10
         DATA 0
SFC      BSS 10                       FILE CHARACTERISTICS BUFFER
```

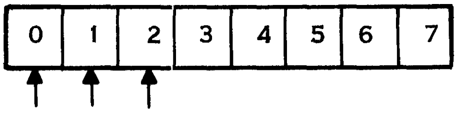
7.3.7 Forward Space

Sub-opcode >06 specifies a Forward Space operation. The Forward Space operation spaces forward over the requested number of logical records, or until an EOF record is encountered. When an EOF record is encountered, file management sets the EOF flag and returns the number of records remaining to be spaced. The file is positioned following the EOF record.

The following fields of the basic supervisor call block apply to a Forward Space operation:

- SVC code — 0
- Return code
- Sub-opcode — >06
- Logical unit number (LUNO)
- System flags
- User flags
- Write character count
- Record number

The following system flags apply to a Forward Space operation:



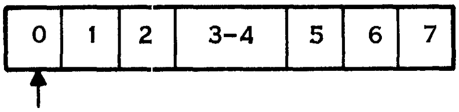
2279594

Bit 0 — Busy flag. Set by system as follows:
1 — Busy.
0 — Operation completed.

Bit 1 — Error flag. Set by system as follows:
1 — Error.
0 — No error.

Bit 2 — End-of-file. Set by system as follows:
1 — EOF record has been read, or physical end of file has been encountered.
0 — EOF record has not been read and physical end of file has not been encountered.

The following user flag applies to a Forward Space operation:



2279595

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

The LUNO field contains the LUNO assigned to the file.

The write character count contains the number of logical records to be forward spaced. File management returns zero when the operation forward spaces the requested number of records without encountering EOF or the physical end of the file. When the Forward Space operation encounters EOF or the physical end of the file, the number of records remaining to be forward spaced is returned.

The following is an example of the source code for a supervisor call block to forward space a file:

```
FSF     DATA 0            FORWARD SPACE FILE ASSIGNED
        BYTE >06,>4C      TO LUNO >4C
        BYTE 0,0
        DATA 0
        DATA 0
        DATA 5            FIVE RECORDS
```

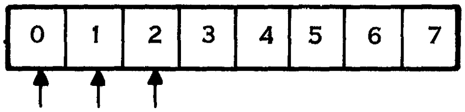
7.3.8 Backward Space

Sub-opcode >07 specifies a Backward Space operation. The Backward Space operation spaces toward the beginning of the file over the requested number of logical records until the beginning of the file or an EOF record is encountered. When an EOF record is encountered, file management sets the EOF flag and returns the number of records remaining to be spaced. The file is positioned to read the EOF record when a Read operation is performed.

The following fields of the basic supervisor call block apply to a Backward Space operation:

- SVC code — 0
- Return code
- Sub-opcode — >07
- Logical unit number (LUNO)
- System flags
- User flags
- Write character count
- Record number

The following system flags apply to a Backward Space operation:



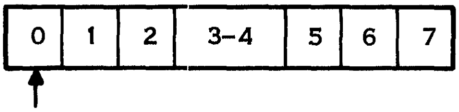
2279596

Bit 0 — Busy flag. Set by system as follows:
1 — Busy.
0 — Operation completed.

Bit 1 — Error flag. Set by system as follows:
1 — .Error.
0 — No error.

Bit 2 — End-of-file. Set by system as follows:
1 — EOF record has been read, or physical beginning of file has been encountered.
0 — EOF record has not been read and physical beginning of file has not been encountered.

The following user flag applies to a Backward Space operation:



2279597

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

The LUNO field contains the LUNO assigned to the file.

The write character count contains the number of logical records to be backward spaced. File management returns zero when the operation spaces backward the requested number of records without encountering EOF or the physical beginning of the file. When the Backward Space operation encounters EOF or the physical beginning of the file, the number of records remaining to be backward spaced is returned.

The following is an example of the source code for a supervisor call block to backward space a file:

```
BSF     DATA 0              BACKWARD SPACE FILE ASSIGNED
        BYTE >07,>4C        TO LUNO >4C
        BYTE 0,0
        DATA 0
        DATA 0
        DATA 3              THREE RECORDS
```

7.3.9 Read ASCII

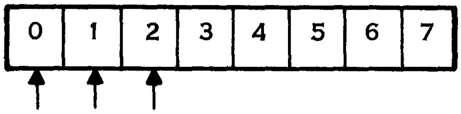
Sub-opcode >09 specifies a Read ASCII operation. The Read ASCII operation reads a record of the file and stores the data in the buffer at the specified address. The characters are packed two per word.

The following fields of the basic supervisor call block apply to a Read ASCII operation:

- SVC code — 0
- Return code
- Sub-opcode — >09
- Logical unit number (LUNO)
- System flags
- User flags
- Data buffer address

- Read character count
- <Actual read count>

The following system flags apply to a Read ASCII operation:



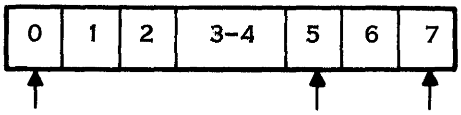
2279598

Bit 0 — Busy flag. Set by system as follows:
1 — Busy.
0 — Operation completed.

Bit 1 — Error flag. Set by system as follows:
1 — Error.
0 — No error.

Bit 2 — End-of-file. Set by system as follows:
1 — EOF record has been read.
0 — EOF record has not been read.

The following user flags apply to a Read ASCII operation:



2279599

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

Bit 5 — Lock/unlock flag. Set as follows:
1 — File management locks the record after reading.
0 — Record remains unlocked.

Bit 7 — Blank adjustment flag. Set as follows:
1 — Fill the buffer with blanks (>20) when the buffer length is greater than the number of characters read.
0 — Do not fill the buffer with blanks.

The LUNO field contains the LUNO assigned to the file.

The data buffer address contains the address of the buffer into which the record is to be read. The address must be an even number.

The read character count contains the maximum number of characters to be read into the buffer.

The actual read count is the number of characters stored in the buffer, returned by file management. The length returned is the length of the record or the length of the buffer, whichever is less. When an odd number of characters is read, an additional character is stored in the buffer, but the odd number (the actual number read) is placed in the actual read count field.

When an EOF record is read, file management returns zero in the actual read count field and sets the EOF flag in the systems flags byte.

When the lock/unlock flag is set, the Read ASCII operation locks the record. The record cannot be read until a Write or Rewrite operation unlocks the record after updating the contents, or until an Unlock operation for the record is performed.

When the blank adjustment flag is set, and the record length is less than the buffer length, file management fills the buffer with blanks. The actual read count contains the buffer length (the total number of characters stored, including blanks) following the operation.

The following is an example of the source code for a supervisor call block to read a file record, and for the required buffer:

```
RASF    DATA 0            READ A RECORD OF FILE ASSIGNED
        BYTE >09,>4C      TO LUNO >4C AND LOCK
        BYTE 0,>04        THE RECORD
        DATA SFRB
        DATA 80
        DATA 0
SFRB    BSS 80            READ BUFFER
```

7.3.10 Write ASCII

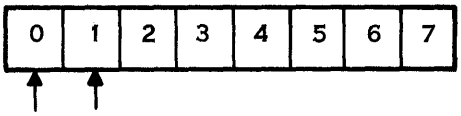
Sub-opcode >0B specifies a Write ASCII operation. The Write ASCII operation transfers the data in the buffer at the specified address to the file. The characters in the buffer are packed two per word.

The following fields of the basic supervisor call block apply to a Write ASCII operation:

- SVC code — 0
- Return code
- Sub-opcode — >0B
- Logical unit number (LUNO)
- System flags
- User flags

- Data buffer address
- Write character count

The following system flags apply to a Write ASCII operation:

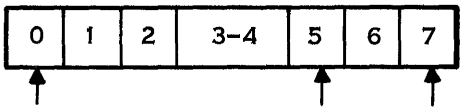


2279600

Bit 0 — Busy flag. Set by system as follows:
1 — Busy.
0 — Operation completed.

Bit 1 — Error flag. Set by system as follows:
1 — Error.
0 — No error.

The following user flags apply to a Write ASCII operation:



2279601

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

Bit 5 — Lock/unlock flag. Set as follows:
1 — File management unlocks the record after writing.
0 — Record remains unlocked.

Bit 7 — Blank adjustment flag. Set as follows:
1 — Do not write trailing blanks in the buffer.
0 — Write the entire buffer contents.

The LUNO field contains the LUNO assigned to the file.

The data buffer address is the address of the buffer that contains the record to be written. The address must be an even number.

The write character count is the number of characters to be written.

A Write ASCII operation clears any EOF indication for the current record or for a subsequent record.

When the lock/unlock flag is set, the Write ASCII operation unlocks the record after the operation completes.

When the blank adjustment flag is set, any trailing blanks in the buffer are not written. That is, the last character of the record actually written is the last nonblank character in the buffer.

The following is an example of the source code for a supervisor call block to write a file record:

```
WASF    DATA 0             WRITE A RECORD TO FILE ASSIGNED
        BYTE >0B,>4C       TO LUNO >4C AND UNLOCK
        BYTE 0,>04         THE RECORD
        DATA SFWB
        DATA 0
        DATA 80
```

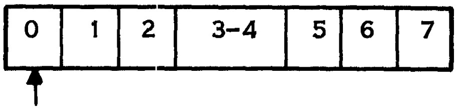
7.3.11 Write EOF

Sub-opcode >0D specifies a Write EOF operation. A Write EOF operation provides an EOF indicator in the file. Any number of EOFs may be written to a sequential file, dividing the file into subfiles.

The following fields of the basic supervisor call block apply to a Write EOF operation:

- SVC code — 0
- Return code
- Sub-opcode — >0D
- Logical unit number (LUNO)
- User flags

The following user flag applies to a Write EOF operation:



2279602

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

The LUNO field contains the LUNO assigned to the file.

The following is an example of the source code for a supervisor call block to write an EOF to a file:

```
WESF     DATA 0                 WRITE AN EOF TO FILE ASSIGNED
         BYTE >0D,>4C           TO LUNO >4C
         BYTE 0,0
         DATA 0
         DATA 0
         DATA 0
```

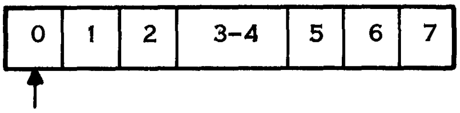
7.3.12 Rewind

Sub-opcode >0E specifies a Rewind operation. The Rewind operation simulates the rewinding of a magnetic tape file. When the file is a sequential file, the next operation performed on the file accesses the first record of the file (not a subfile).

The following fields of the basic supervisor call block apply to a Rewind operation:

- SVC code — 0
- Return code
- Sub-opcode — >0E
- Logical unit number (LUNO)
- User flags

The following user flag applies to a Rewind operation:



2279603

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

The LUNO field contains the LUNO assigned to the file.

The following is an example of the source code for a supervisor call block to rewind a file:

```
RWSF     DATA 0                 REWIND FILE ASSIGNED TO LUNO >4C
         BYTE >0E,>4C
         BYTE 0,0
         DATA 0
         DATA 0
         DATA 0
```

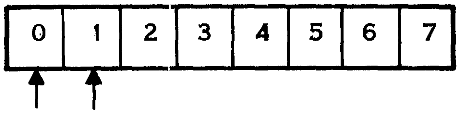
7.3.13 Rewrite

Sub-opcode >10 specifies a Rewrite operation. The Rewrite operation backspaces a file one logical record, and writes a record to replace the record previously read. The write portion of the operation is similar to the Write ASCII operation.

The following fields of the basic supervisor call block apply to a Rewrite operation:

- SVC code — 0
- Return code
- Sub-opcode — >10
- Logical unit number (LUNO)
- System flags
- User flags
- Data buffer address
- Write character count

The following system flags apply to a Rewrite operation:

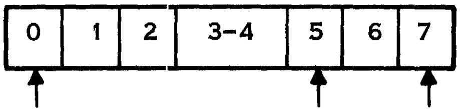


2279604

Bit 0 — Busy flag. Set by system as follows:
1 — Busy.
0 — Operation completed.

Bit 1 — Error flag. Set by system as follows:
1 — Error.
0 — No error.

The following user flags apply to a Rewrite operation:



2279605

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

Bit 5 — Lock/unlock flag. Set as follows:
1 — File management unlocks the record after rewriting.
0 — Record remains in the state it was in before the rewrite.

Bit 7 — Blank adjustment flag. Set as follows:
1 — Do not write trailing blanks in the buffer.
0 — Write the entire buffer contents.

The LUNO field contains the LUNO assigned to the file.

The data buffer address is the address of the buffer that contains the record to be written. The address must be an even number.

The write character count is the number of characters to be written.

When the character count indicates that the length of the updated record is different from the length of the record in the file, the record is not rewritten and the operation terminates in error. A blank compressed record may be rewritten, but the length of the new record (with blank compression) must remain the same as the length of the blank compressed record in the file.

A Rewrite operation does not alter the end-of-file. When a rewrite is attempted on the end-of-file record, the rewritten record is lost.

The following is an example of the source code for a supervisor call block to rewrite a record:

```
RWTSF   DATA 0              REWRITE RECORD OF FILE ASSIGNED
        BYTE >10,>4C        TO LUNO 4C
        BYTE 0,0
        DATA SFWB
        DATA 0
        DATA 80
```

7.3.14 Modify Access Privileges

Sub-opcode >11 specifies a Modify Access Privileges operation. The Modify Access Privileges operation assigns access privileges to a file. The requested access privileges are not allowed if those access privileges to the file are currently in use. In that case, the existing access privileges continue to apply.

The following fields of the basic supervisor call block apply to a Modify Access Privileges operation:

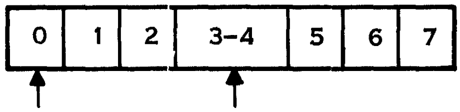
- SVC code — 0

- Return code

- Sub-opcode — >11

- Logical unit number (LUNO)
- User flags

The following user flags apply to a Modify Access Privileges operation:



2279606

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

Bits 3-4 — Access privilege flag. Set as follows:
00 — Exclusive write.
01 — Exclusive all.
10 — Shared.
11 — Read only.

The LUNO field contains the LUNO assigned to the file.

The access privilege flag in the user flags byte specifies the new access privileges for the file.

The following is an example of the source code for a supervisor call block to modify the access privileges of a file:

```
MASF     DATA 0                     MODIFY ACCESS PRIVILEGES OF FILE
         BYTE >11,>4C               ASSIGNED TO LUNO 4C TO
         BYTE 0,>18                 READ ONLY
         DATA 0
         DATA 0
         DATA 0
```

7.3.15 Open Extend

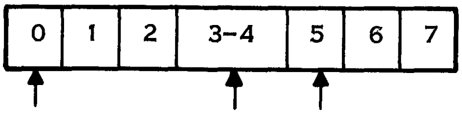
Sub-opcode >12 specifies an Open Extend operation. The Open Extend operation opens a file and positions the file at the EOF record that follows the last data record in the file. For a file with a single EOF, the file is positioned at that EOF record. For a file with multiple EOFs, the file is positioned at the first EOF of the group. Except for positioning the file, the Open Extend operation is the same as the Open operation previously described.

The following fields of the basic supervisor call block apply to an Open Extend operation:

- SVC code — 0
- Return code
- Sub-opcode — >12

- Logical unit number (LUNO)
- User flags
- Data buffer address
- Read character count

The following user flags apply to an Open Extend operation:



2279607

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

Bits 3-4 — Access privilege flag. Set as follows:
00 — Exclusive write.
01 — Exclusive all.
10 — Shared.
11 — Read only.

Bit 5 — Do not replace flag. Applies to shared, exclusive write, and exclusive all access only. Set as follows:
1 — Open file that was created with the autocreate option when the LUNO was assigned. When file existed prior to the Assign LUNO operation, terminate in error.
0 — Open file regardless of when it was created.

The LUNO field contains the LUNO assigned to the file to be opened.

The Open Extend operation returns the file type code in the buffer address field. The file type code for a sequential file is >01FF.

When the calling task places zero in the input character count field, the Open Extend operation returns the logical record length specified for the file.

The following is an example of the source code for a supervisor call block to open a sequential file to be extended:

```
OXSF      DATA 0                       OPEN EXTEND FILE ASSIGNED TO LUNO
          BYTE >12,>4C                 >4C WITH SHARED ACCESS
          BYTE 0,>10
XSFT      DATA 0
XBL       DATA 0
          DATA 0
```

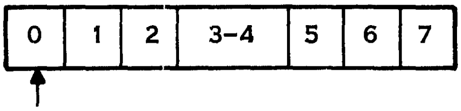
7.3.16 Unlock

Sub-opcode >4A specifies an Unlock operation. The Unlock operation releases exclusive control of the current record, whether the record was locked by another task or by the calling task.

The following fields of the basic supervisor call block apply to an Unlock operation:

- SVC code — 0
- Return code
- Sub-opcode — >4A
- Logical unit number (LUNO)
- User flags

The following user flag applies to an Unlock operation:



2279608

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

The LUNO field contains the LUNO assigned to the file.

The following is an example of the source code for a supervisor call block to unlock a file record:

```
USF     DATA 0              UNLOCK CURRENT RECORD IN FILE ASSIGNED
        BYTE >4A,>4C        TO LUNO >4C
        BYTE 0,0
        DATA 0
        DATA 0
        DATA 0
```

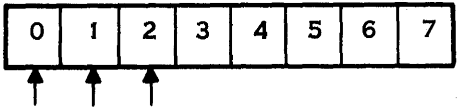
7.3.17 Multiple Record Read

Sub-opcode >59 specifies a Multiple Record Read operation. The Multiple Record Read operation reads an integral number of records of the file and stores the data in the buffer at the specified address. The characters are packed one per byte. The calling task specifies the number of characters to be stored; the operation stores complete records, each preceded by a word that contains the length of the record. The transfer of data to the buffer continues until the number of characters in the current record exceeds the number of characters remaining in the buffer.

The following fields of the basic supervisor call block apply to a Multiple Record Read operation:

- SVC code — 0
- Return code
- Sub-opcode — >59
- Logical unit number (LUNO)
- System flags
- User flags
- Data buffer address
- Read character count
- <Actual read count>

The following system flags apply to a Multiple Record Read operation:



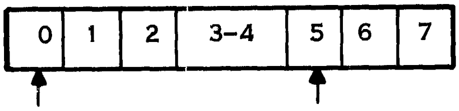
2279609

Bit 0 — Busy flag. Set by system as follows:
1 — Busy.
0 — Operation completed.

Bit 1 — Error flag. Set by system as follows:
1 — Error.
0 — No error.

Bit 2 — End-of-file. Set by system as follows:
1 — EOF record has been read.
0 — EOF record has not been read.

The following user flags apply to a Multiple Record Read operation:



2279610

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

Bit 5 — Lock/unlock flag. Set as follows:
1 — File management locks the records after reading.
0 — Records remain unlocked.

The LUNO field contains the LUNO assigned to the file.

The data buffer address contains the address of the buffer in which the records are to be stored. The address must be an even number.

The read character count contains the maximum number of characters to be stored in the buffer.

The actual read count is the actual number of characters stored, returned by file management. The operation reads a record and compares the number of characters in the record with the remaining buffer space. When the record can be stored in the remaining space, the operation stores the record length (in characters) in a word, followed by the characters of the record. The operation continues reading and storing records until the record length is greater than the remaining space. At this point, the operation returns the total number of characters in the records plus two for the overhead word of each record and terminates.

If the buffer is too small to hold one record, an error code is returned.

When the first record read is an EOF record, file management returns zero in the actual read count length field and sets the EOF flag in the systems flags byte. When an EOF record is read in a subsequent record, the operation terminates and returns the number of characters stored without setting the EOF flag. The next Read operation sets the EOF flag.

When the lock/unlock flag.is set, the Multiple Record Read operation locks the records that are read. These records cannot be read again until a Write or Rewrite operation unlocks each record after updating the contents, or until an Unlock operation for each record is performed.

When a Multiple Record Read operation reads a record that is locked, the operation returns an error message and no more records are read. The contents of any unlocked records read prior to reading the locked record are stored in the data buffer, and the actual read count contains the number of characters stored in the buffer.

If an end of medium is reached, a >0030 error is returned and the EOF flag is not set.

The following is an example of the source code for a supervisor call block for a Multiple Record Read to read ten 80-character records and for the required buffer:

```
MRRSF    DATA 0              READ MULTIPLE RECORDS OF FILE
         BYTE >59,>4C        ASSIGNED TO LUNO >4C
         BYTE 0,0
         DATA MRB
         DATA 820
         DATA 0
MRB      BSS 820             READ BUFFER
```

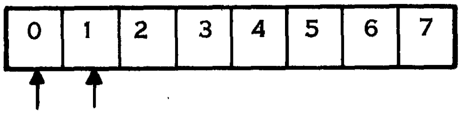
7.3.18 Multiple Record Write

Sub-opcode >5B specifies a Multiple Record Write operation. The Multiple Record Write operation transfers the data in the buffer at the specified address to the file. The characters in the buffer are packed one per byte. The first character of each record in the buffer must be at an even (word) address and must be preceded by a word that contains the record length (in characters). The record length words are not written to the file.

The following fields of the basic supervisor call block apply to a Multiple Record Write operation:

- SVC code — 0
- Return code
- Sub-opcode — >5B
- Logical unit number (LUNO)
- System flags
- User flags
- Data buffer address
- Write character count

The following system flags apply to a Multiple Record Write operation:

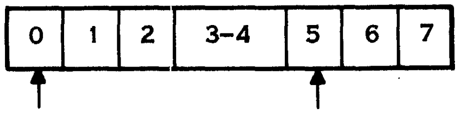


2279611

Bit 0 — Busy flag. Set by system as follows:
1 — Busy.
0 — Operation completed.

Bit 1 — Error flag. Set by system as follows:
1 — Error.
0 — No error.

The following user flags apply to a Multiple Record Write operation:



2279612

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

Bit 5 — Lock/unlock flag. Set as follows:
1 — File management unlocks the records after writing.
0 — Records remain in the state they were in before the write.

The LUNO field contains the LUNO assigned to the file.

The data buffer address is the address of the buffer that contains the records to be written. A word that contains the length (in characters) precedes each record.

The write character count is the number of characters to be written. This count must include the overhead words containing the record lengths.

A Multiple Record Write operation clears any EOF indication for the current record or for a subsequent record.

When the lock/unlock flag is set, all locked records that are written are unlocked following the Multiple Record Write operation.

The following is an example of the source code for a supervisor call block for a Multiple Record Write operation and for a typical buffer:

```
MRWSF    DATA 0                    WRITE MULTIPLE RECORDS TO FILE
         BYTE >5B,>4C              ASSIGNED TO LUNO >4C
         BYTE 0,0
         DATA SFWB
         DATA 0
         DATA ENDB-SFWB
SFWB     DATA REC2-$               WRITE BUFFER
         TEXT 'RECORD 1'
           .
           .
           .
REC2     DATA REC3-$
         TEXT 'RECORD 2'
           .
           .
           .
REC3     DATA ENDB-$
         TEXT 'RECORD 3'
           .
           .
           .
         EVEN
ENDB     EQU $
```

7.4 RELATIVE RECORD FILE I/O

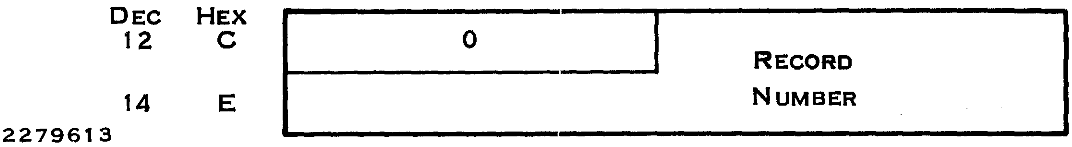
Relative record file I/O uses the basic supervisor call block previously shown to effect I/O transfers, file positioning, and other I/O operations. Each record of a relative record file is designated by a record number. The first record is record 0; the maximum record number is 16,777,215. A 4-byte extension contains the record number. The user must set the first byte to 0. The last three bytes are used as follows:

SVC >00 -- I/O OPERATIONS

ALIGN ON WORD BOUNDARY
CAN BE INITIATED AS AN EVENT

DEC	HEX		
0	0	>00	<RETURN CODE>
2	2	SUB-OPCODE	LUNO
4	4	<SYSTEM FLAGS>	USER FLAGS
6	6	DATA BUFFER ADDRESS	
8	8	READ CHARACTER COUNT	
10	A	WRITE CHARACTER COUNT/<ACTUAL READ COUNT>	

2279470



Byte	Contents
12	Set to zero.
13–15	Record number to which operation applies.

The sub-opcodes that apply to relative record files are:

00	Open
01	Close
02	Close, Write Logical EOF
03	Open and Rewind
04	Close and Unload
05	Read File Characteristics
06	Forward Space
07	Backward Space
09	Read ASCII
0B	Write ASCII
0D	Write Logical EOF
0E	Rewind
10	Rewrite
11	Modify Access Privileges
12	Open Extend
4A	Unlock Record
59	Multiple Record Read
5B	Multiple Record Write

The following sub-opcodes perform operations identical to those shown:

Sub-opcode	Operation	Identical to
0A	Read Direct	Read ASCII
0C	Write Direct	Write ASCII

The following sub-opcodes are ignored:

08	Not used
0F	Unload

Except for the Read File Characteristics operation, the file must be opened using sub-opcode >00, >03, or >12 prior to any I/O operation.

7.4.1 Open

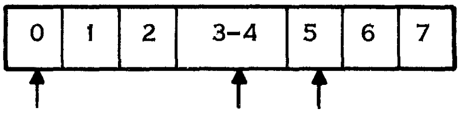
Sub-opcode >00 specifies an Open operation. The Open operation enables the task to perform I/O operations on the file assigned to the LUNO. If the Open operation is successful, the access privilege requested in bits 3 and 4 of byte 5 is granted to the task. An Open operation must be performed before a task can perform any I/O operation except a Read File Characteristics operation.

The following fields of the basic call block apply to an Open operation:

- SVC code — 0
- Return code
- Sub-opcode — >00
- Logical unit number (LUNO)
- User flags
- Data buffer address
- Read character count

The following user flags apply to an Open operation:

2279614



Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

Bits 3-4 — Access privilege flag. Set as follows:
00 — Exclusive write.
01 — Exclusive all.
10 — Shared.
11 — Read only.

Bit 5 — Do not replace flag. Applies to shared, exclusive write, and exclusive all access only. Set as follows:
1 — Open file that was created with the autocreate option when the LUNO was assigned. When file existed prior to the Assign LUNO operation, terminate in error.
0 — Open file regardless of when it was created.

The LUNO field contains the LUNO assigned to the file to be opened.

The Open operation returns the file type code in the buffer address field. The file type codes for relative record files are:

>02FF	Relative record file, no special usage
>04FF	Directory (relative record) file
>05FF	Program (relative record) file
>06FF	Image (relative record) file

When the calling task places zero in the input character count field, the Open operation returns the logical record length specified for the file.

The following is an example of the source code for a supervisor call block to open a relative record file:

```
ORRF    DATA 0             OPEN FILE ASSIGNED TO LUNO >4E
        BYTE 0,>4E         WITH EXCLUSIVE ALL ACCESS
        BYTE 0,>08
RRFT    DATA 0
BLR     DATA 0
        DATA 0
        DATA 0
        DATA 0
```

7.4.2 Close

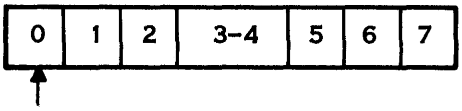
Sub-opcode >01 specifies a Close operation. The Close operation ends I/O to a LUNO from the calling task. The LUNO remains assigned to the file. Specifically, for the file assigned to the LUNO, the Close operation:

- Unlocks any locked records
- Writes all modified file blocks on which write was deferred
- Releases access privileges
- Updates file data structures maintained by the system to accurately describe the current file. Until the Close operation is performed, data structures on disk do not accurately reflect the contents of the file. If a system crash occurs before a Close is performed, new records written to the end of an existing file will be lost.

The following fields of the basic supervisor call block apply to a Close operation:

- SVC code — 0
- Return code
- Sub-opcode — >01
- Logical unit number (LUNO)
- User flags

The following user flag applies to a Close operation:



2279615

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

The LUNO field contains the LUNO assigned to the file to be closed.

The following is an example of the source code for a supervisor call block to close a file:

```
CRRF     DATA 0                  CLOSE FILE ASSIGNED TO LUNO >4E
         BYTE >01,>4E
         BYTE 0,0
         DATA 0
         DATA 0
         DATA 0
         DATA 0
         DATA 0
```

7.4.3 Close, Write Logical EOF

Sub-opcode >02 specifies a Close, Write Logical EOF operation. A Close, Write Logical EOF operation consists of a Write Logical EOF operation followed by a Close operation.

7.4.4 Open and Rewind

Sub-opcode >03 specifies an Open and Rewind operation. An Open and Rewind operation performs an Open operation followed by a Rewind operation.

7.4.5 Close and Unload

Sub-opcode >04 specifies a Close and Unload operation. The Close and Unload operation is the same as a Close operation.

7.4.6 Read File Characteristics

Sub-opcode >05 specifies a Read File Characteristics operation. The Read File Characteristics operation returns file characteristics information in a buffer specified by the user. The file characteristics consist of 10 bytes of information.

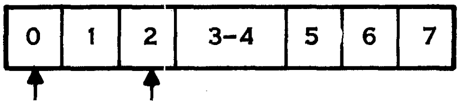
In a secure environment, this operation may be used to determine which access rights a user has to a file. If this option is used, a word of file access rights is returned.

The following fields of the basic supervisor call block apply to a Read File Characteristics operation:

- SVC code — 0
- Return code

- Sub-opcode — >05
- Logical unit number (LUNO)
- User flags
- Data buffer address
- Read character count
- <Actual read count>

The following user flag applies to a Read File Characteristics operation:



2279616

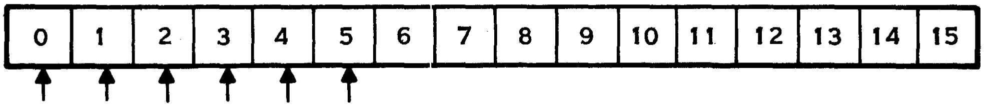
Bit 0 — Initiate flag. Set as follows.
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.
Bit 2 — Security access rights flag. Set as follows:
1 — System returns a word of rights in the buffer specified by the user.
0 — System returns file characteristics.

The LUNO field contains the LUNO assigned to the file for which characteristics are to be read.

The data buffer address is the address of the buffer into which DNOS places the file characteristics. The buffer should contain at least 10 bytes if the security rights option is not used. If the security rights option is used, the buffer should contain two bytes.

The read character count is the length of the buffer.

If the security access rights flag is set, a word of file access rights is returned to the buffer specified by the user. The following explains the meaning of the bits in the returned word:



2285028

Bit 0 — 1 if the user has read access
Bit 1 — 1 if the user has write acces
Bit 2 — 1 if the user has delete access
Bit 3 — 1 if the user has execute access
Bit 4 — 1 if the user has control access

If the security access rights flag is not set, 10 bytes of file characteristics are returned.

DNOS returns the number of characters stored in the buffer in the actual read count field. The file characteristics for a relative record file (other than a program file) consist of 10 characters. Two additional characters are required for program files. The contents of the buffer following a Read File Characteristics operation are:

DEC	HEX		
0	0	FILE ATTRIBUTE FLAGS	
2	2	PHYSICAL RECORD LENGTH	
4	4	LOGICAL RECORD LENGTH	
6	6	NUMBER OF LOGICAL RECORDS	
8	8		
10	A	SECTORS/BLOCK	SECTORS/ADU

2279617

Byte **Contents**

0-1 File attribute flags, as follows:

Bits 0-1 — File usage:
00 — No special usage.
01 — Directory file.
10 — Program file.
11 — Image file.

Bits 2-3 — Data format:
00 — Not blank compressed.
01 — Blank compressed.

Bit 4 — Allocation type:
0 — Fixed size file.
1 — Expandable file.

Bits 5-6 — File type:
10 — Relative record.

Bit 7 — Write protection flag:
0 — Not write protected.
1 — Write protected.

Bit 8 — Delete protection flag:
0 — Not delete protected.
1 — Delete protected.

Bit 9 — Temporary file flag:
0 — Permanent file.
1 — Temporary file.

Bit 10 — Blocked file flag:
0 — Blocked.
1 — Not blocked.

Bit 11 — Reserved.

Bit 12 — Write mode flag:
0 — Deferred write.
1 — Immediate write.

Bits 13-15 — Reserved.

2-3 Physical record length.

4-5 Logical record length.

6-9 Number of logical records.

Program files only:

10 Sectors/block.

11 Sectors/ADU, varies with type of disk.

The following is an example of the source code for a supervisor call block to read file characteristics, and for the required buffer:

```
RRRFC    DATA 0              READ CHARACTERISTICS OF FILE ASSIGNED
         BYTE >05,>4E        TO LUNO >4E
         BYTE 0,0
         DATA RRFC
         DATA 12
         DATA 0
         DATA 0
         DATA 0
RRFC     BSS 12              FILE CHARACTERISTICS BUFFER
```

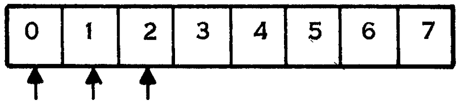
7.4.7 Forward Space

Sub-opcode 06 specifies a Forward Space operation. The Forward Space operation spaces forward over the requested number of logical records, or until the end of the file is encountered. When the end of the file is encountered, file management sets the logical EOF flag and returns the number of records remaining to be spaced.

The following fields of the basic supervisor call block and the relative record file extension apply to a Forward Space operation:

- SVC code — 0
- Return code
- Sub-opcode — >06
- Logical unit number (LUNO)
- System flags
- User flags
- Write character count

The following system flags apply to a Forward Space operation:



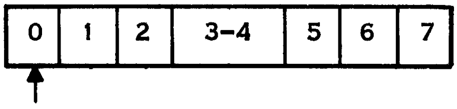
2279618

Bit 0 — Busy flag. Set by system as follows:
1 — Busy.
0 — Operation completed.

Bit 1 — Error flag. Set by system as follows:
1 — Error.
0 — No error.

Bit 2 — End-of-file. Set by system as follows:
1 — Physical end-of-file has been encountered.
0 — Physical end-of-file has not been encountered.

The following user flag applies to a Forward Space operation:



2279619

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

The LUNO field contains the LUNO assigned to the file.

The write character count contains the number of logical records to be forward spaced. File management returns zero when the operation forward spaces the requested number of records without encountering the physical end of the file. When the Forward Space operation encounters the physical end of the file, the number of records remaining to be forward spaced is returned.

The following is an example of the source code for a supervisor call block to forward space a file:

```
FRRF    DATA 0              FORWARD SPACE FILE ASSIGNED
        BYTE >06,>4E        TO LUNO >4E
        BYTE 0,0
        DATA 0
        DATA 0
        DATA 5              FIVE RECORDS
        DATA 0
        DATA 0
```

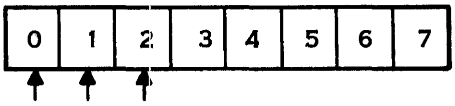
7.4.8 Backward Space

Sub-opcode >07 specifies a Backward Space operation. The Backward Space operation spaces toward the beginning of the file over the requested number of logical records, or until the beginning of the file is encountered. When the beginning of the file is encountered, file management sets the logical EOF flag and returns the number of records remaining to be spaced. The file is positioned to read the last record spaced over when a Read operation is performed.

The following fields of the basic supervisor call block and the relative record file extension apply to a Backward Space operation:

- SVC code — 0
- Return code
- Sub-opcode — >07
- Logical unit number (LUNO)
- System flags
- User flags
- Write character count

The following system flags apply to a Backward Space operation:



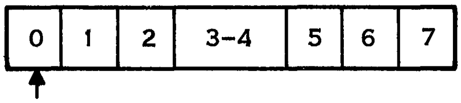
2279620

Bit 0 — Busy flag. Set by system as follows:
1 — Busy.
0 — Operation completed.

Bit 1 — Error flag. Set by system as follows:
1 — Error.
0 — No error.

Bit 2 — End-of-file. Set by system as follows:
1 — Physical beginning of file has been encountered.
0 — Physical beginning of file has not been encountered.

The following user flag applies to a Backward Space operation:



2279621

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

The LUNO field contains the LUNO assigned to the file.

The write character count contains the number of logical records to be backward spaced. File management returns zero when the operation spaces backward the requested number of records without encountering the physical beginning of the file. When the Backward Space operation encounters the physical beginning of the file, the number of records remaining to be backward spaced is returned.

The following is an example of the source code for a supervisor call block to backward space a file:

```
BRRF      DATA 0                    BACKWARD SPACE FILE ASSIGNED
          BYTE >07,>4E              TO LUNO >4E
          BYTE 0,0
          DATA 0
          DATA 0
          DATA 3                    THREE RECORDS
          DATA 0
          DATA 0
```

7.4.9 Read ASCII

Sub-opcode >09 specifies a Read ASCII operation. The Read ASCII operation reads a record of the file and stores the data in the buffer at the specified address. The characters are packed one per byte.

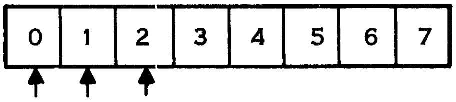
The following fields of the basic supervisor call block and the relative record file extension apply to a Read ASCII operation:

- SVC code — 0

- Return code

- Sub-opcode — >09
- Logical unit number (LUNO)
- System flags
- User flags
- Data buffer address
- Read character count
- <Actual read count>
- Record number

The following system flags apply to a Read ASCII operation:



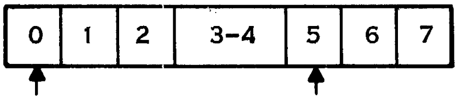
2279622

Bit 0 — Busy flag. Set by system as follows:
1 — Busy.
0 — Operation completed.

Bit 1 — Error flag. Set by system as follows:
1 — Error.
0 — No error.

Bit 2 — Logical end-of-file. Set by system as follows:
1 — Logical EOF record has been read.
0 — Logical EOF record has not been read.

The following user flags apply to a Read ASCII operation:



2279623

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

Bit 5 — Lock/unlock flag. Set as follows:
1 — File management locks the record after reading.
0 — Record remains in the state it was in before the read.

The LUNO field contains the LUNO assigned to the file.

The data buffer address contains the address of the buffer into which the record is to be read. The address must be an even number.

The read character count contains the maximum number of characters to be read into the buffer.

The actual read count is the number of characters read, returned by file management. The length returned is the length of the record or the length of the buffer, whichever is less.

The record number field contains the number of the record to be read. File management increments this number by one when a Read ASCII operation completes successfully.

When the record number just past the last record is read, file management returns zero in the input record length field and sets the logical EOF flag in the system flags byte.

When the lock/unlock flag is set, the Read ASCII operation locks the record. The record cannot be read until a Write or Rewrite operation unlocks the record after updating the contents, or until an Unlock operation for the record is performed.

The following is an example of the source code for a supervisor call block to read a file record, and for the required buffer:

```
RARRF    DATA 0                  READ A RECORD OF FILE ASSIGNED
         BYTE >09,>4E            TO LUNO >4E AND LOCK
         BYTE 0,>04              THE RECORD
         DATA RRFRB
         DATA 80
         DATA 0
         DATA 0                  RECORD NUMBER
         DATA 35
SFRB     BSS 80                  READ BUFFER
```

7.4.10 Write ASCII

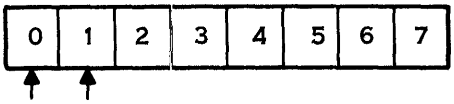
Sub-opcode >0B specifies a Write ASCII operation. The Write ASCII operation transfers the data in the buffer at the specified address to the file. The characters in the buffer are packed two per word.

The following fields of the basic supervisor call block and the relative record file extension apply to a Write ASCII operation:

- SVC code — 0
- Return code
- Sub-opcode — >0B
- Logical unit number (LUNO)
- System flags

- User flags
- Data buffer address
- Write character count
- Record number

The following system flags apply to a Write ASCII operation:

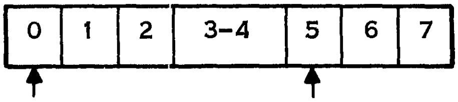


2279624

Bit 0 — Busy flag. Set by system as follows:
1 — Busy.
0 — Operation completed.

Bit 1 — Error flag. Set by system as follows:
1 — Error.
0 — No error.

The following user flags apply to a Write ASCII operation:



2279625

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

Bit 5 — Lock/unlock flag. Set as follows:
1 — File management unlocks the record after writing.
0 — Record remains in the state it was in before the write.

The LUNO field contains the LUNO assigned to the file.

The data buffer address is the address of the buffer that contains the record to be written. The address must be an even number.

The write character count is the number of characters in the buffer. The number of characters to be written is the logical record length of the file. When the buffer contains fewer characters than a logical record, the system writes the specified number of characters and fills the record with zeros. When the buffer contains more characters than the logical record, the system writes the data, truncating the data at the logical record length.

The record number field contains the number of the record to be written. File management increments this number by one when a Write ASCII operation completes successfully.

When a Write ASCII operation writes a record with a record number that is the highest numbered record in the file, the next higher numbered record becomes the end-of-file record; that is, the end-of-file record is not affected by a Write operation unless a record with a record number equal to or greater than that of the end-of-file record is written.

Within the limits of file expandability, writing a record with a record number higher than that of the end-of-file expands the file to include the new record. The contents of the records skipped over in writing the new record are not altered; they contain whatever happens to be on the disk.

When the lock/unlock flag is set, the Write ASCII operation unlocks the record after the Write operation.

The following is an example of the source code for a supervisor call block to write a file record:

```
WARRF    DATA 0             WRITE A RECORD TO FILE ASSIGNED
         BYTE >0B,>4E       TO LUNO >4E AND UNLOCK
         BYTE 0,>04         THE RECORD
         DATA SFWB
         DATA 0
         DATA 80
         DATA 0             RECORD NUMBER
         DATA 75
```

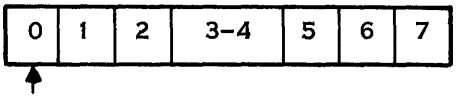
7.4.11 Write Logical EOF

Sub-opcode >0D specifies a Write Logical EOF operation. A Write Logical EOF operation stores the record number in the call block for the Write Logical EOF operation as the end-of-file.

The following fields of the basic supervisor call block and the relative record file extension apply to a Write EOF operation:

- SVC code — 0
- Return code
- Sub-opcode — >0D
- Logical unit number (LUNO)
- User flags
- Record number

The following user flag applies to a Write Logical EOF operation:



2279626

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

The LUNO field contains the LUNO assigned to the file.

The record number field contains the number of the record that becomes the logical EOF record.

The following is an example of the source code for a supervisor call block to write a logical EOF on a relative record file:

```
WEFRRF  DATA 0               DESIGNATE RECORD 85 AS EOF
        BYTE >0D,>4E         OF FILE ASSIGNED TO LUNO >4E
        BYTE 0,0
        DATA 0
        DATA 0
        DATA 0
        DATA 0               RECORD NUMBER
        DATA 85
```

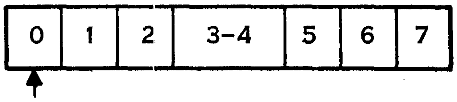
7.4.12 Rewind

Sub-opcode >0E specifies a Rewind operation. The Rewind operation simulates the rewinding of a magnetic tape file. For a relative record file, the operation stores zero in the record number field.

The following fields of the basic supervisor call block and the relative record file extension apply to a Rewind operation:

- SVC code — 0
- Return code
- Sub-opcode — >0E
- Logical unit number (LUNO)
- User flags
- Record number

The following user flag applies to a Rewind operation:



2279627

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

The LUNO field contains the LUNO assigned to the file.

The following is an example of the source code for a supervisor call block to rewind a file:

```
RWRRF    DATA 0                  REWIND FILE ASSIGNED TO LUNO >4E
         BYTE >0E,>4E
         BYTE 0,0
         DATA 0
         DATA 0
         DATA 0
         DATA 0                  RECORD NUMBER —
         DATA 345                SET TO ZERO BY REWIND
```

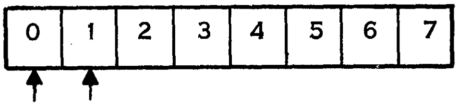
7.4.13 Rewrite

Sub-opcode >10 specifies a Rewrite operation. The Rewrite operation backspaces a file one logical record, and writes a record to replace the record previously read. The write portion of the operation is similar to the Write ASCII operation.

The following fields of the basic supervisor call block and the relative record file extension apply to a Rewrite operation:

- SVC code — 0
- Return code
- Sub-opcode — >10
- Logical unit number (LUNO)
- System flags
- User flags
- Data buffer address
- Write character count
- Record number

The following system flags apply to a Rewrite operation:

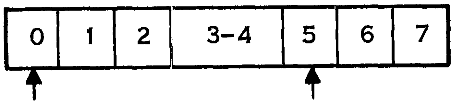


2279628

Bit 0 — Busy flag. Set by system as follows:
1 — Busy.
0 — Operation completed.

Bit 1 — Error flag. Set by system as follows:
1 — Error.
0 — No error.

The following user flags apply to a Rewrite operation:



2279629

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

Bit 5 — Lock/unlock flag. Set as follows:
1 — File management locks the record after reading.
0 — Record remains in the state it was in before the rewrite.

The LUNO field contains the LUNO assigned to the file.

The data buffer address is the address of the buffer that contains the record to be written. The address must be an even number.

The write character count is the number of characters in the buffer. The number of characters to be written is the logical record length of the file. When the buffer contains fewer characters than a logical record, the system writes the specified number of characters and fills the record with zeros. When the buffer contains more characters than the logical record, the system writes the data, truncating the data at the logical record length.

The record number field contains a record number one greater than the number of the record to be rewritten.

When a Rewrite is attempted on the end-of-file record number, the record is written and the end-of-file record number is increased by one.

The following is an example of the source code for a supervisor call block to rewrite a record:

```
RWTRRF   DATA 0                    REWRITE RECORD OF FILE ASSIGNED
         BYTE >10,>4E              TO LUNO 4E
         BYTE 0,0
         DATA SFWB
         DATA 0
         DATA 80
         DATA 0
         DATA 954
```

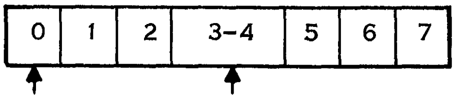
7.4.14 Modify Access Privileges

Sub-opcode >11 specifies a Modify Access Privileges operation. The Modify Access Privileges operation assigns access privileges to a file. When the requested access privileges are not allowed, the existing access privileges continue to apply.

The following fields of the basic supervisor call block apply to a Modify Access Privileges operation:

- SVC code — 0
- Return code
- Sub-opcode — >11
- Logical unit number (LUNO)
- User flags

The following user flags apply to a Modify Access Privileges operation:



2279630

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

Bits 3-4 — Access privilege flag. Set as follows:
00 — Exclusive write.
01 — Exclusive all.
10 — Shared.
11 — Read only.

The LUNO field contains the LUNO assigned to the file.

The access privilege flag in the user flags byte specifies the new access privileges for the file.

The following is an example of the source code for a supervisor call block to modify the access privileges of a file:

```
MARRF    DATA 0                 MODIFY ACCESS PRIVILEGES OF FILE
         BYTE >11,>4E           ASSIGNED TO LUNO 4E TO
         BYTE 0,0               EXCLUSIVE WRITE
         DATA 0
         DATA 0
         DATA 0
         DATA 0
         DATA 0
```

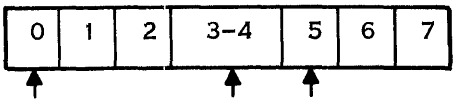
7.4.15 Open Extend

Sub-opcode >12 specifies an Open Extend operation. The Open Extend operation for a relative record file is effectively an Open operation except that the file is positioned at the logical EOF record. That is, the record number is set to the number of the logical EOF record.

The following fields of the basic supervisor call block apply to an Open Extend operation:

- SVC code — 0
- Return code
- Sub-opcode — >12
- Logical unit number (LUNO)
- User flags
- Data buffer address
- Read character count
- Record number

The following user flags apply to an Open Extend operation:



2279631

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

Bits 3-4 — Access privilege flag. Set as follows:
00 — Exclusive write.
01 — Exclusive all.
10 — Shared.
11 — Read only.

Bit 5 — Do not replace flag. Applies to shared, exclusive write, and exclusive all access only. Set as follows:
1 — Open file that was created with the autocreate option when the LUNO was assigned. When file existed prior to the Assign LUNO operation, terminate in error.
0 — Open file regardless of when it was created.

The LUNO field contains the LUNO assigned to the file to be opened.

The Open Extend operation returns the file type code in the buffer address field. The file type codes for relative record files are:

>02FF	Relative record file, no special usage
>04FF	Directory (relative record) file
>05FF	Program (relative record) file
>06FF	Image (relative record) file

When the calling task places zero in the read character count field, the Open Extend operation returns the logical record length specified for the file.

The following is an example of the source code for a supervisor call block to open a relative record file with an Open Extend operation:

```
OXRRF    DATA 0               OPEN EXTEND FILE ASSIGNED TO LUNO
         BYTE >12,>4E         >4E WITH SHARED ACCESS
         BYTE 0,>10
XRRFT    DATA 0
XBLR     DATA 0
         DATA 0
         DATA 0
         DATA 0
```

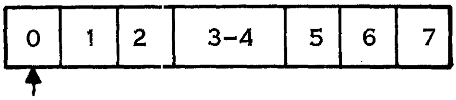
7.4.16 Unlock

Sub-opcode >4A specifies an Unlock operation. The Unlock operation releases exclusive control of any previously locked record, a record locked by another task, or a record locked by the calling task. For a relative record file, the operation unlocks a specified record.

The following fields of the basic supervisor call block and the relative record file extension apply to an Unlock operation:

- SVC code — 0
- Return code
- Sub-opcode — >4A
- Logical unit number (LUNO)
- User flags
- Record number

The following user flag applies to an Unlock operation:



2279632

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

The LUNO field contains the LUNO assigned to the file.

The following is an example of the source code for a supervisor call block to unlock a file record:

```
URRF    DATA 0              UNLOCK RECORD 456 IN FILE
        BYTE >4A,>4E        ASSIGNED TO LUNO >4E
        BYTE 0,0
        DATA 0
        DATA 0
        DATA 0
        DATA 0
        DATA 456
```

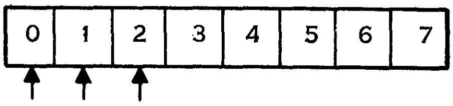
7.4.17 Multiple Record Read

Sub-opcode >59 specifies a Multiple Record Read operation. The Multiple Record Read operation reads an integral number of records of the file and stores the data in the buffer at the specified address. The characters are packed two per word. The calling task specifies the number of characters to be stored; the operation stores complete records, each preceded by a word that contains the length of the record. The transfer of data to the buffer continues until the number of characters in the current record exceeds the number of characters remaining in the buffer. The specified record of a relative record file is the first record to be read.

The following fields of the basic supervisor call block and the relative record file extension apply to a Multiple Record Read operation:

- SVC code — 0
- Return code
- Sub-opcode — >59
- Logical unit number (LUNO)
- System flags
- User flags
- Data buffer address
- Read character count
- <Actual read count>
- Record number

The following system flags apply to a Multiple Record Read operation:



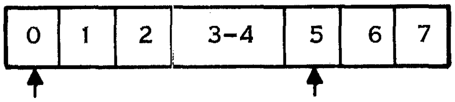
2279633

Bit 0 — Busy flag. Set by system as follows:
1 — Busy.
0 — Operation completed.

Bit 1 — Error flag. Set by system as follows:
1 — Error.
0 — No error.

Bit 2 — End-of-file. Set by system as follows:
1 — EOF record has been read.
0 — EOF record has not been read.

The following user flags apply to a Multiple Record Read operation:



2279634

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

Bit 5 — Lock/unlock flag. Set as follows:
1 — File management locks the records after reading.
0 — Records remain in the state they were in before the read.

The LUNO field contains the LUNO assigned to the file.

The data buffer address contains the address of the buffer in which the records are to be stored. The address must be an even number.

The input character count contains the maximum number of characters to be stored in the buffer.

The actual read count is the number of characters stored; it is returned by file management. The operation reads a record, beginning at the record specified in the record number field, and compares the number of characters in the record with the remaining buffer space. When the record can be stored in the remaining space, the operation stores the record length (in characters) in a word, followed by the characters of the record. The operation continues reading and storing records until the record length is greater than the remaining space. At this point, the operation returns the total number of characters in the records plus two for the overhead word of each record and terminates.

If the buffer supplied is too small, an error code is returned.

When the first record read is a logical EOF record, file management returns zero in the actual read count field and sets the logical EOF flag in the system flags byte. When a logical EOF record is read in a subsequent record, the operation terminates and returns the number of characters stored without setting the logical EOF flag. The next read operation sets the logical EOF flag.

When the lock/unlock flag is set, the Multiple Record Read operation locks the records that are read. These records cannot be read again until a Write or Rewrite operation unlocks each record after updating the contents, or until an Unlock operation for each record is performed.

When a Multiple Record Read operation reads a record that is locked, the operation returns an error message and no more records are read. The contents of any unlocked records read prior to reading the locked record are stored in the data buffer, and the actual read count contains the number of characters stored in the buffer.

Reading begins at the record specified in the record number field. The record number is incremented by one as each record is read. At the completion of the operation, the record number field contains the number of the record following the last record read.

The following is an example of the source code for a supervisor call block for a Multiple Record Read, and for the required buffer:

```
MRRRRF    DATA 0                        READ MULTIPLE RECORDS OF FILE
          BYTE >59,>4E                  ASSIGNED TO LUNO >4E
          BYTE 0,0
          DATA MRRB
          DATA 800
          DATA 0
          DATA 0
          DATA 38
MRRB      BSS                           READ BUFFER
```

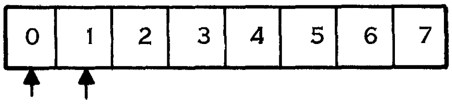
7.4.18 Multiple Record Write

Sub-opcode >5B specifies a Multiple Record Write operation. The Multiple Record Write operation transfers the data in the buffer at the specified address to the file. The characters in the buffer are packed two per word. The first character of each record in the buffer is preceded by a word that contains the record length (in characters). The record length words are not written to the file.

The following fields of the basic supervisor call block and the relative record file extension apply to a Multiple Record Write operation:

- SVC code — 0
- Return code
- Sub-opcode — >5B
- Logical unit number (LUNO)
- System flags
- User flags
- Data buffer address
- Write character count
- Record number

The following system flags apply to a Multiple Record Write operation:

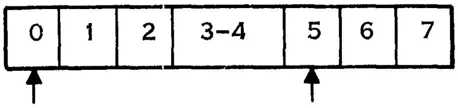


2279635

Bit 0 — Busy flag. Set by system as follows:
1 — Busy.
0 — Operation completed.

Bit 1 — Error flag. Set by system as follows:
1 — Error.
0 — No error.

The following user flags apply to a Multiple Record Write operation:



2279636

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

Bit 5 — Lock/unlock flag. Set as follows:
1 — File management unlocks the records after writing.
0 — Records remains unlocked.

The LUNO field contains the LUNO assigned to the file.

The data buffer address is the address of the buffer that contains the records to be written. A word that contains the length (in characters) precedes each record.

The write character count is the number of characters in the buffer, including the overhead words containing the record lengths. The number of characters to be written is a multiple of the logical record length of the file. When the record length supplied in the buffer for a record is less than the logical record length, the system writes the specified number of characters and fills the record with zeros. When the record length is greater than the logical record length, the system writes the data, truncating the data at the logical record length.

The record number is the number of the first record to be written. The system increments the record number by one as each record is written. At the completion of the operation, the record number field contains the number of the record following the last record written. The record following the highest-numbered record in the file is the logical EOF record.

When the lock/unlock flag is set, all records that are written are unlocked following the Multiple Record Write operation.

The following is an example of the source code for a supervisor call block for a Multiple Record Write operation:

```
WMRRRF   DATA 0                      WRITE RECORDS TO FILE
         BYTE >5B,>4E                ASSIGNED TO LUNO >4E
         BYTE 0,0
         DATA RRFWB
         DATA 0
         DATA ENDBF-RRFWB
         DATA 0
         DATA 248
RRFWB    DATA RREC2-$                WRITE BUFFER
         TEXT 'RECORD 248'
           .
           .
           .
RREC2    DATA RREC3-$
         TEXT 'RECORD 249'
           .
           .
           .
RREC3    DATA ENDBF-$
         TEXT 'RECORD 250'
           .
           .
           .
         EVEN
ENDBF    EQU  $
```

7.5 KEY INDEXED FILE I/O

DNOS supports resource-independent operations to key indexed files. These operations access the file as if it were a sequential file with records written in the order of the primary key. The resource-specific operations access records in the sequence of any key defined for the file, or access a specific record by its key. The supported operations read, insert, locate, or rewrite a record.

7.5.1 Key Indexed File Resource-Independent I/O

Key indexed file I/O uses the basic supervisor call block previously shown to effect I/O transfers, file positioning, and other I/O operations. Except for the Read File Characteristics operation, the file must be opened for resource-independent I/O using sub-opcode >00 or >03. Sub-opcode >01 (Close) is common to both resource-independent and resource-specific operations. The sub-opcodes that apply to key indexed file resource-independent I/O are:

SVC >00 -- I/O OPERATIONS

ALIGN ON WORD BOUNDARY
CAN BE INITIATED AS AN EVENT

DEC	HEX		
0	0	>00	<RETURN CODE>
2	2	SUB-OPCODE	LUNO
4	4	<SYSTEM FLAGS>	USER FLAGS
6	6	DATA BUFFER ADDRESS	
8	8	READ CHARACTER COUNT	
10	A	WRITE CHARACTER COUNT/<ACTUAL READ COUNT>	

2279470

00	Open
01	Close
03	Open and Rewind
05	Read File Characteristics
06	Forward Space
07	Backward Space
09	Read ASCII
0E	Rewind

Sub-opcode >0A, Read Direct, is identical to a Read ASCII operation.

The following sub-opcodes return error messages:

02	Close, Write EOF
04	Close and Unload
08	Not Used
0B	Write ASCII
0C	Write Direct
0D	Write EOF
0F	Unload

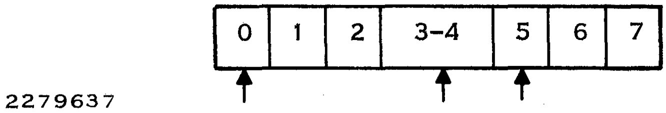
7.5.1.1 Open. Sub-opcode >00 specifies an Open operation for resource-independent I/O. The Open operation enables the task to perform I/O operations on the file assigned to the LUNO. If the Open operation is successful, the access privilege requested in bits 3 and 4 of byte 5 is granted to the task. An Open operation must be performed before a task can access a file. However, the Read File Characteristics operation may be performed without opening the file. The first Open operation for the LUNO positions the file at the first record (lowest-valued primary key). Subsequent Open operations for the same LUNO open the file as positioned by the most recent operation.

The following fields of the basic call block apply to an Open operation:

- SVC code — 0
- Return code

- Sub-opcode — >00
- Logical unit number (LUNO)
- User flags
- Data buffer address
- Read character count

The following user flags apply to an Open operation:



Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

Bits 3-4 — Access privilege flag. Set as follows:
00 — Exclusive write.
01 — Exclusive all.
10 — Shared.
11 — Read only.

Bit 5 — Do not replace flag. Applies to shared, exclusive write, and exclusive all access only. Set as follows:
1 — Open file that was created with the autocreate option when the LUNO was assigned. When file existed prior to the Assign LUNO operation, terminate in error.
0 — Open file regardless of when it was created.

The LUNO field contains the LUNO assigned to the file to be opened.

The Open operation returns the file type code in the buffer address field. The file type code for a key indexed file is >03FF.

When the calling task places zero in the read character count field, the Open operation returns the logical record length specified for the file.

The following is an example of the source code for a supervisor call block to open a key indexed file:

```
OKIF     DATA 0              OPEN FILE ASSIGNED TO LUNO >5E
         BYTE 0,>5E          WITH SHARED ACCESS
         BYTE 0,>10
KFT      DATA 0
BLK      DATA 0
         DATA 0
```

7.5.1.2 Close. Sub-opcode >01 specifies a Close operation. The Close operation ends I/O to a LUNO from the calling task. The LUNO remains assigned to the file. Specifically, for the file assigned to the LUNO, the Close operation:

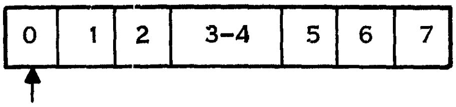
- Unlocks any locked records
- Writes all modified file blocks on which write was deferred
- Releases access privileges
- Updates file data structures maintained by the system to accurately describe the current file. Until the Close operation is performed, data structures on disk do not accurately reflect the contents of the file. If a system crash occurs before a Close is performed, new records written to the end of an existing file will be lost.

The Close operation is used to close files opened either for resource-independent or for resource-specific operations.

The following fields of the basic supervisor call block apply to a Close operation:

- SVC code — 0
- Return code
- Sub-opcode — >01
- Logical unit number (LUNO)
- User flags

The following user flag applies to a Close operation:



2279638

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

The LUNO field contains the LUNO assigned to the file to be closed.

The following is an example of the source code for a supervisor call block to close a file:

```
CKIF     DATA 0                CLOSE FILE ASSIGNED TO LUNO >5E
         BYTE >01,>5E
         BYTE 0,0
         DATA 0
         DATA 0
         DATA 0
```

7.5.1.3 Open and Rewind. Sub-opcode >03 specifies an Open and Rewind operation. An Open and Rewind operation is identical to an Open operation except that the Open and Rewind operation always positions the file to the record having the lowest-valued primary key (first record).

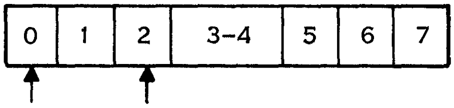
7.5.1.4 Read File Characteristics. Sub-opcode >05 specifies a Read File Characteristics operation. The Read File Characteristics operation returns file characteristics information in a buffer specified by the user. The file characteristics consist of 12+4n bytes of information, in which n is the number of keys defined for the file, 14 or fewer.

In a secure environment, this operation may be used to determine which access rights a user has to a file. If this option is used, a word of file access rights is returned.

The following fields of the basic supervisor call block apply to a Read File Characteristics operation:

- SVC code — 0
- Return code
- Sub-opcode — >05
- Logical unit number (LUNO)
- User flags
- Data buffer address
- Read character count
- <Actual read count>

The following user flag applies to a Read File Characteristics operation:



2279639

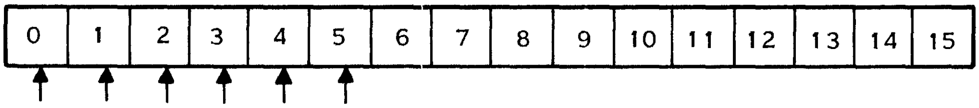
Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.
Bit 2 — Security access rights flag. Set as follows:
1 — System returns a word of rights in the buffer specified by the user.
0 — System returns file characteristics.

The LUNO field contains the LUNO assigned to the file for which characteristics are to be read.

The data buffer address is the address of the buffer into which DNOS places the file characteristics. The buffer should contain at least 16 bytes (for a file with a primary key only) if the security rights option is not used. If the security rights option is used, the buffer should contain two bytes. Add four bytes for each secondary key. The buffer should contain at least 10 bytes if the security rights option is not used. If the security rights option is used, the buffer should contain two bytes.

The read character count is the length of the buffer.

If the security access rights flag is set, a word of file access rights is returned to the buffer specified by the user. If the security access rights flag is not set, file characteristics are returned. The following explains the meaning of the bits in the returned word:

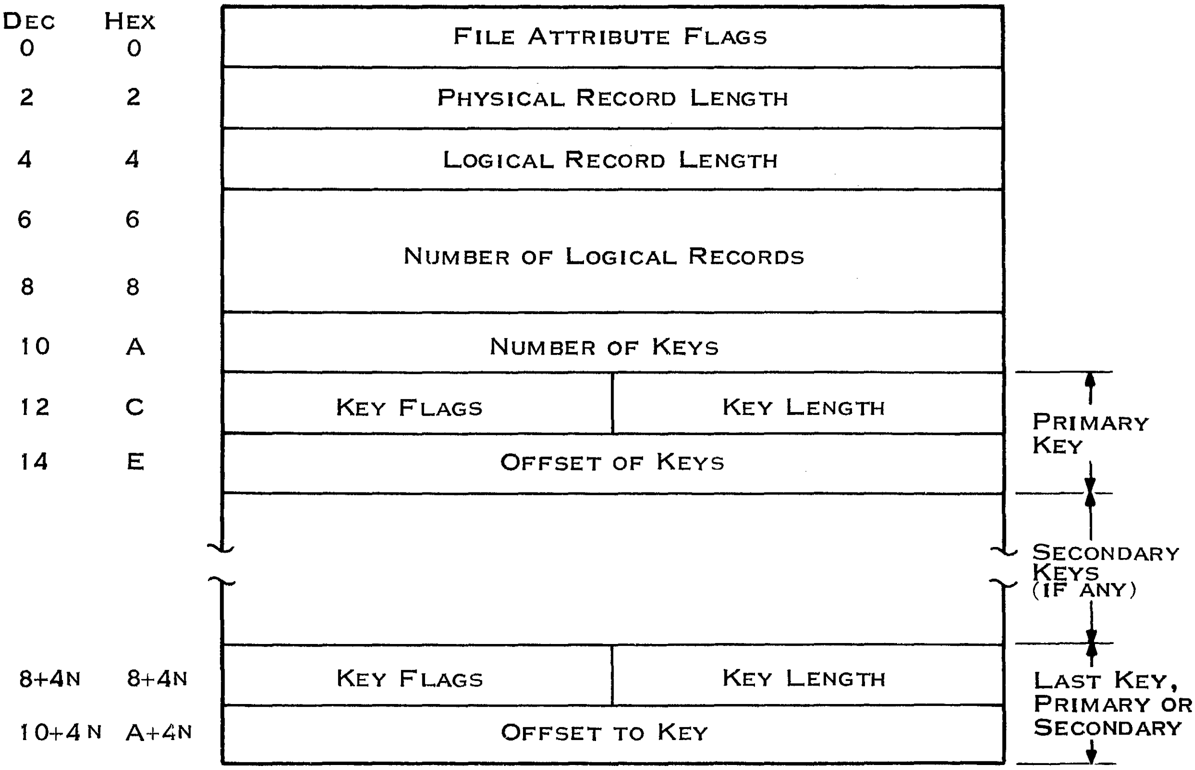


2285029

Bit 0 — 1 if the user has read access
Bit 1 — 1 if the user has write access
Bit 2 — 1 if the user has delete access
Bit 3 — 1 if the user has execute access
Bit 4 — 1 if the user has control access

If the security access rights flag is not set, a number of file characteristics is returned.

DNOS returns the number of characters stored in the buffer in the actual read count field. The file characteristics for a key indexed file consist of 16 through 68 characters. The contents of the buffer following a Read File Characteristics operation are:



2279640

Byte	Contents
0-1	File attribute flags, as follows:

Bits 0-1 — File usage:
00 — No special usage.

Bits 2-3 — Data format:
01 — Blank compressed.

Bit 4 — Allocation type:
1 — Expandable file.

Bits 5-6 — File type:
11 — Key indexed.

Bit 7 — Write protection flag:
0 — Not write protected.
1 — Write protected.

Byte	Contents
	Bit 8 — Delete protection flag: 0 — Not delete protected. 1 — Delete protected.
	Bit 9 — Temporary file flag: 0 — Permanent file. 1 — Temporary file.
	Bit 10 — Blocked file flag: 0 — Blocked.
	Bit 11 — Reserved.
	Bit 12 — Write mode flag: 0 — Deferred write. 1 — Immediate write.
	Bits 13-15 — Reserved.
2-3	Physical record length.
4-5	Logical record length.
6-9	Number of logical records.
10-11	Number of keys for the file, 1 through 14.
For each of n keys:	
8+4n	Key flags, as follows: Bits 0-4 — Reserved. Bit 5 — Modifiable flag. Set to 1 if key may be modified, or to 0 if key may not be modified. Bit 6 — Reserved. Bit 7 — Duplicatable flag. Set to 1 if key may be duplicated, or to 0 if key may not be duplicated.
9+4n	Number of characters in key.
10+4n	Offset to key — The character position in the logical record where the key begins (first position is 0).

The following is an example of the source code for a supervisor call block to read file characteristics and for the buffer required for a three-key file :

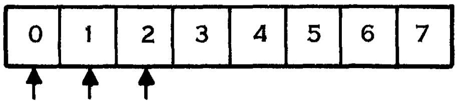
```
RKIFC     DATA 0                  READ CHARACTERISTICS OF FILE
          BYTE >05,>5E            ASSIGNED TO LUNO >5E
          BYTE 0,0
          DATA KIFC
          DATA 24
          DATA 0
KIFC      BSS 24                  FILE CHARACTERISTICS BUFFER
```

7.5.1.5 Forward Space. Sub-opcode >06 specifies a Forward Space operation. The Forward Space operation spaces forward (in primary key order) over the requested number of logical records, or until the logical EOF is encountered. When the EOF is encountered, file management sets the EOF flag and returns the number of records remaining to be spaced. The file is positioned before the EOF.

The following fields of the basic supervisor call block apply to a Forward Space operation:

- SVC code — 0
- Return code
- Sub-opcode — >06
- Logical unit number (LUNO)
- System flags
- User flags
- Write character count

The following system flags apply to a Forward Space operation:



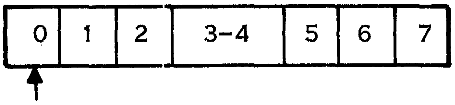
2279641

Bit 0 — Busy flag. Set by system as follows:
1 — Busy.
0 — Operation completed.

Bit 1 — Error flag. Set by system as follows:
1 — Error.
0 — No error.

Bit 2 — End-of-file. Set by system as follows:
1 — EOF has been encountered.
0 — EOF has not been encountered.

The following user flag applies to a Forward Space operation:



2279642

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

The LUNO field contains the LUNO assigned to the file.

The write character count contains the number of logical records to be forward spaced. File management returns zero when the operation forward spaces the requested number of records without encountering EOF. When the forward space operation encounters EOF, the number of records remaining to be forward spaced is returned.

The following is an example of the source code for a supervisor call block to forward space a file:

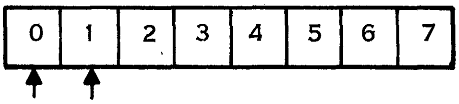
```
FSKIF     DATA 0                    FORWARD SPACE FILE ASSIGNED
          BYTE >06,>5E              TO LUNO >5E
          BYTE 0,0
          DATA 0
          DATA 0
          DATA 5                    FIVE RECORDS
```

7.5.1.6 Backward Space. Sub-opcode >07 specifies a Backward Space operation. The Backward Space operation spaces toward the beginning of the file (decreasing primary key order) over the requested number of logical records, or until the beginning of the file is encountered. When the beginning of the file is encountered, file management returns the number of records remaining to be spaced. The file is positioned to read the first record when a Read operation is performed.

The following fields of the basic supervisor call block apply to a Backward Space operation:

- SVC code — 0
- Return code
- Sub-opcode — >07
- Logical unit number (LUNO)
- System flags
- User flags
- Write character count

The following system flags apply to a Backward Space operation:

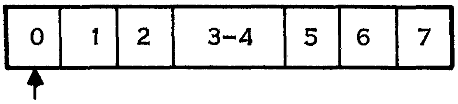


2279643

Bit 0 — Busy flag. Set by system as follows:
1 — Busy.
0 — Operation completed.

Bit 1 — Error flag. Set by system as follows:
1 — Error.
0 — No error.

The following user flag applies to a Backward Space operation:



2279644

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

The LUNO field contains the LUNO assigned to the file.

The write character count contains the number of logical records to be backward spaced. File management returns zero when the operation spaces backward the requested number of records without encountering the beginning of the file. When the backward space operation encounters the beginning of the file, the number of records remaining to be backward spaced is returned.

The following is an example of the source code for a supervisor call block to backward space a file:

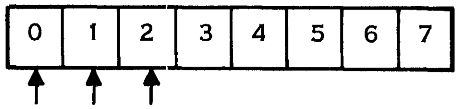
```
BSKIF     DATA 0               BACKWARD SPACE FILE ASSIGNED
          BYTE >07,>5E         TO LUNO >5E
          BYTE 0,0
          DATA 0
          DATA 0
          DATA 3               THREE RECORDS
```

7.5.1.7 Read ASCII. Sub-opcode >09 specifies a Read ASCII operation. The Read ASCII operation reads a record of the file and stores the data in the buffer at the specified address. The characters are packed two per word.

The following fields of the basic supervisor call block apply to a Read ASCII operation:

- SVC code — 0
- Return code
- Sub-opcode — >09
- Logical unit number (LUNO)
- System flags
- User flags
- Data buffer address
- Input character count
- <Input record length>

The following system flags apply to a Read ASCII operation:



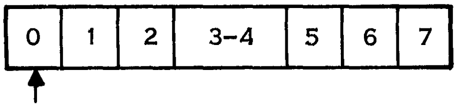
2279645

Bit 0 — Busy flag. Set by system as follows:
1 — Busy.
0 — Operation completed.

Bit 1 — Error flag. Set by system as follows:
1 — Error.
0 — No error.

Bit 2 — End-of-file. Set by system as follows:
1 — EOF has been encountered.
0 — EOF has not been encountered.

The following user flag applies to a Read ASCII operation:



2279646

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

The LUNO field contains the LUNO assigned to the file.

The data buffer address contains the address of the buffer into which the record is to be read. The address must be an even number.

The read character count contains the maximum number of characters to be read into the buffer.

The actual read count is the number of characters read, returned by file management. The length returned is the length of the record or the length of the buffer, whichever is less. When an odd number of characters is read, an additional character is stored in the buffer, but the odd number (the actual number read) is placed in the actual read count field.

When an EOF is encountered, file management returns zero in the actual read count field and sets the EOF flag in the system flags byte.

The following is an example of the source code for a supervisor call block to read a file record, and for the required buffer:

```
RKIF     DATA 0            READ A RECORD OF FILE ASSIGNED
         BYTE >09,>5E      TO LUNO >5E
         BYTE 0,0
         DATA KIFRB
         DATA 80
         DATA 0
KIFRB    BSS 80            READ BUFFER
```

7.5.1.8 Rewind. Sub-opcode >0E specifies a Rewind operation. The Rewind operation simulates the rewinding of a magnetic tape file. The operation positions the file at the first record, that is, at the record having the lowest-valued primary key.

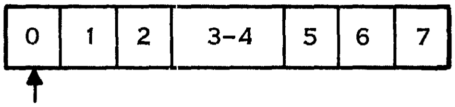
The following fields of the basic supervisor call block apply to a Rewind operation:

- SVC code — 0
- Return code
- Sub-opcode — >0E

- Logical unit number (LUNO)

- User flags

The following user flag applies to a Rewind operation:



2279647

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

The LUNO field contains the LUNO assigned to the file.

The following is an example of the source code for a supervisor call block to rewind a file:

```
RWKIF     DATA 0                        REWIND FILE ASSIGNED TO LUNO >5E
          BYTE >0E,>5E
          BYTE 0,0
          DATA 0
          DATA 0
          DATA 0
```

7.5.1.9 Modify Access Privileges. Sub-opcode >11 specifies a Modify Access Privileges operation. The Modify Access Privileges operation assigns access privileges to a file. When the requested access privileges are not allowed, the existing access privileges continue to apply.

The following fields of the basic supervisor call block apply to a Modify Access Privileges operation:

- SVC opcode — >00

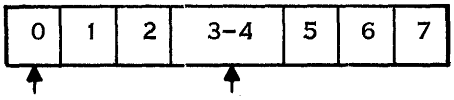
- Return code

- Sub-opcode — >11

- Logical unit number (LUNO)

- User flags

The following user flags apply to a Modify Access Privileges operation:



2279630

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

Bits 3-4 — Access privilege flag. Set as follows:
00 — Exclusive write.
01 — Exclusive all.
10 — Shared.
11 — Read only.

The LUNO field contains the LUNO assigned to the file.

The access privilege flag in the user flags byte specifies the new access privileges for the file.

The following is an example of the source code for a supervisor call block to modify the access privileges of a file:

```
MARRF    DATA 0               MODIFY ACCESS PRIVILEGES OF FILE
         BYTE >11,>4E         ASSIGNED TO LUNO 4E TO
         BYTE 0,0             EXCLUSIVE WRITE
         DATA 0
         DATA 0
         DATA 0
         DATA 0
         DATA 0
```

7.5.2 Key Indexed File Resource-Specific I/O

DNOS supports 15 operations that exploit the capabilities of key indexed files, as follows:

40	Open Random
41	Read Greater
42	Read by Key
42	Read Current
44	Read Greater or Equal
45	Read Next
46	Insert
47	Rewrite
48	Read Previous
49	Delete by Key
49	Delete Current
4A	Unlock
50	Set Currency Equal
51	Set Currency Greater or Equal
52	Set Currency Greater

The key indexed file must be opened using the Open Random operation before performing any of the other operations listed. The Close operation described for resource-independent key indexed file I/O applies also to files opened using the Open Random operation.

Some of the resource-specific operations require a key to specify the record on which the operation is to be performed. Other operations use information returned by file management during a previous operation to specify the record. Operations that require a key are:

Read by Key
Read Greater
Read Greater or Equal
Set Currency Equal
Set Currency Greater
Set Currency Greater or Equal
Delete by Key

Operations that use the information supplied to a previous operation are:

Read Current
Read Previous
Read Next
Rewrite
Unlock
Delete Current

Several of the operations that specify a record by its key can use a partial key to specify a record. A partial key consists of fewer characters than the key defined for the record. The partial key begins with the first character of the defined key. A partial key may not be taken from the center or end of the defined key. For example, if a key consists of a telephone number (area code, exchange code, and number) the area code (first three digits) or the area code and exchange code (first six digits) are valid partial keys. The exchange code (fourth through sixth digits) is not a valid partial key. The following operations may use a partial key:

Read Greater
Read Greater or Equal
Set Currency Equal
Set Currency Greater
Set Currency Greater or Equal

The Set Currency operations locate a record by returning information that a subsequent command may use to read or delete a record.

The key indexed file resource specific operations use a single-word extension to the basic supervisor call block, as follows:

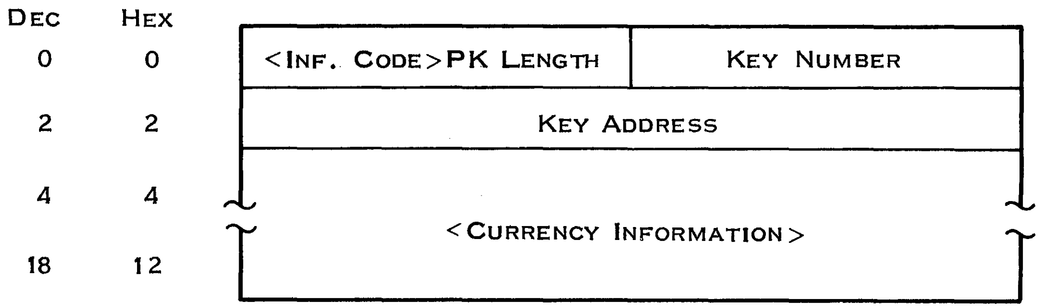
DEC	HEX	
12	C	CURRENCY BLOCK ADDRESS

2279648

Byte	Contents
12	Address of the currency block (must be a word boundary address).

To access the last record of a key indexed file, several operations are needed. Use the Set Currency Greater or Equal operation, specifying a key value larger than that for any record in the file. This operation returns an informative code indicating that the key does not exist. Use the currency returned by this operation and do a Read Previous operation. The Read will return the last record of the file.

The currency block contains the address of a block that contains the key and an area in which file management returns information that enables access to the record by commands that do not supply a key. The structure of the currency block is:



2279649

Byte	Contents
0	System returns informative code. For partial key operations, length of partial key.
1	Number of key (order of definition).
2–3	Address of block that contains the key or the partial key. The address must be an even number.
4–19	Currency information returned by file management. This information may be used by a subsequent operation to access the same record without the key.

The key or partial key for the operation is in a block or buffer. The address of the block is placed in the currency block.

All resource-specific operations except Open Random and Unlock return an informative code in byte 0 of the currency block. The informative code is a code that gives the status of an operation. The informative codes are listed in Table 7-1; they are not necessarily error codes, but they indicate some abnormal condition resulting from the operation.

Table 7-1. Key Indexed File Informative Codes

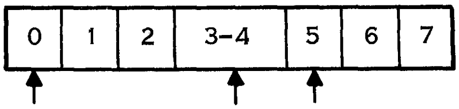
Code (Hexadecimal)	Meaning
00	Normal completion.
B3	No more records to be read.
B4	Additional records with same key value as that specified for the current read operation.
B5	No record with specified key or currency information.
B7	Record to be locked is already locked, or record to be deleted is locked.
B8	Record specified by sub-opcode and currency information does not exist.
BD	Next record for Read Next or Read Previous operation cannot be found. Either the record has been deleted or currency information has been destroyed.

7.5.2.1 Open Random. Sub-opcode >40 specifies an Open Random operation for a key indexed file. The Open Random operation enables the task to perform resource-specific I/O operations to the file assigned to the LUNO. If the Open operation is successful, the access privilege requested in bits 3 and 4 of byte 5 is granted to the task. An Open Random operation must be performed before a task can access a key indexed file for a resource-specific operation. The Open Random operation does not alter the position of the file.

The following fields of the basic call block apply to an Open Random operation:

- SVC code — 0
- Return code
- Sub-opcode — >40
- Logical unit number (LUNO)
- User flags
- Data buffer address
- Read character count

The following user flags apply to an Open Random operation:



2279650

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

Bits 3-4 — Access privilege flag. Set as follows:
00 — Exclusive write.
01 — Exclusive all.
10 — Shared.
11 — Read only.

Bit 5 — Do not replace flag. Applies to shared, exclusive write, and exclusive all access only. Set as follows:
1 — Open file that was created with the autocreate option when the LUNO was assigned. When file existed prior to the Assign LUNO operation, terminate in error.
0 — Open file regardless of when it was created.

The LUNO field contains the LUNO assigned to the file to be opened.

The Open Random operation returns the file type code in the buffer address field. The file type code for a key indexed file is >03FF.

When the calling task places zero in the read character count field, the Open Random operation returns the logical record length specified for the file.

The following is an example of the source code for a supervisor call block for an Open Random operation:

```
ORKIF    DATA 0                  OPEN FILE ASSIGNED TO LUNO >5E
         BYTE >40,>5E            WITH SHARED ACCESS
         BYTE 0,>10
ORKFT    DATA 0
ORBLK    DATA 0
         DATA 0
```

7.5.2.2 Read by Key. Sub-opcode >42 specifies a Read by Key operation. The Read by Key operation reads a record of a key indexed file that contains the specified key and stores the data in the buffer at the specified address. The characters are packed two per word. When the file contains more than one record having the specified key, the operation reads the record that was inserted in the file first.

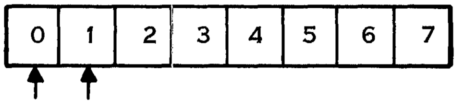
The following fields of the supervisor call block as extended for key indexed files apply to a Read by Key operation:

- SVC code — 0
- Return code
- Sub-opcode — >42
- Logical unit number (LUNO)
- System flags
- User flags
- Data buffer address
- Read character count
- <Actual read count>
- Currency block address

The following fields of the currency block apply to a Read by Key operation:

- <Informative code>
- Key number
- Key address

The following system flags apply to a Read by Key operation:

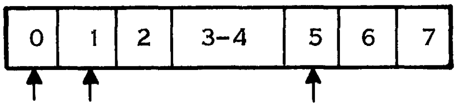


2279651

Bit 0 — Busy flag. Set by system as follows:
1 — Busy.
0 — Operation completed.

Bit 1 — Error flag. Set by system as follows:
1 — Error.
0 — No error.

The following user flags apply to a Read by Key operation:



2279652

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

Bit 1 — Key specified flag. Set to one for a Read by Key operation. (The Read Current operation uses the same sub-opcode with this flag set to 0.)

Bit 5 — Lock/unlock flag. Set as follows:
1 — File management locks the record after reading.
0 — Record remains in the state it was in before the read.

The LUNO field contains the LUNO assigned to the file.

The data buffer address contains the address of the buffer into which the record is to be read. The address must be an even number.

The read character count contains the maximum number of characters to be read into the buffer.

The actual read count is the actual number of characters read, returned by file management. The count returned is the length of the record or the length of the buffer, whichever is less. When an odd number of characters is read, an additional character is stored in the buffer, but the odd number (the actual number read) is placed in the actual read count field.

The key number is the number of the key as defined when the file was created. The MKF command shows the numbers of the keys of a key indexed file.

The key address is the address of the buffer that contains the key of the requested record. The key need not be in a buffer used exclusively by the Read by Key operation. It may be shared by various portions of the program. The length of the key is defined when the file is created.

A Read by Key operation may return >B4 or >B5 as the informative code. When the lock/unlock flag is set to one, the operation may also return >B7 as the informative code.

The following is an example of the source code to read a file record specified by key. The supervisor call block, read buffer, currency block, and key block are as follows:

```
RBK       DATA 0                         READ A RECORD OF FILE ASSIGNED
          BYTE >42,>5E                   TO LUNO >5E
          BYTE 0,>40
          DATA RBKRB
          DATA 80
          DATA 0
          DATA CRBK
RBKRB     BSS 80                         READ BUFFER
CRBK      BYTE 0                         CURRENCY BLOCK
          BYTE 3                         KEY NUMBER 3
          DATA RBKEY
CINRBK    BSS 16                         ASSUMES KEY LENGTH IS 5
RBKEY     TEXT 'TEXAS'
```

7.5.2.3 Read Greater. Sub-opcode >41 specifies a Read Greater operation. The Read Greater operation reads the first record of a key indexed file that contains a value greater than the specified value for a specified key and stores the data in the buffer at the specified address. The characters are packed two per word. When the file contains more than one record having the requested key value, the operation reads the record that was inserted in the file first.

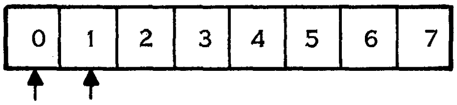
The following fields of the supervisor call block as extended for key indexed files apply to a Read Greater operation:

- SVC code — 0
- Return code
- Sub-opcode — >41
- Logical unit number (LUNO)
- System flags
- User flags
- Data buffer address
- Read character count
- <Actual read count>
- Currency block address

The following fields of the currency block apply to a Read Greater operation:

- <Informative code>
- Length of partial key
- Key number
- Key address

The following system flags apply to a Read Greater operation:

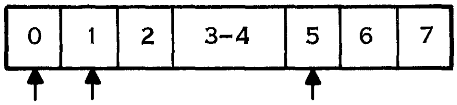


2279653

Bit 0 — Busy flag. Set by system as follows:
1 — Busy.
0 — Operation completed.

Bit 1 — Error flag. Set by system as follows:
1 — Error.
0 — No error.

The following user flags apply to a Read Greater operation:



2279654

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

Bit 1 — Key specified flag. Set to one for a Read Greater operation.

Bit 5 — Lock/unlock flag. Set as follows:
1 — File management locks the record after reading.
0 — Record remains in the state it was in before the read.

The LUNO field contains the LUNO assigned to the file.

The data buffer address contains the address of the buffer into which the record is to be read. The address must be an even number.

The read character count contains the maximum number of characters to be read into the buffer.

The actual read count is the number of characters read, returned by file management. The count returned is the length of the record or the length of the buffer, whichever is less. When an odd number of characters is read, an additional character is stored in the buffer, but the odd number (the actual number read) is placed in the actual read count field.

The length of the partial key field contains the length of the partial key for operations to which a partial key applies. The field must be set to zero or to the defined length when the defined length of the key applies.

The key number is the number of the key as defined when the file was created. The MKF command displays the numbers of the keys of a key indexed file.

The key address is the address of the buffer that contains the key of the requested record. The key need not be in a buffer used exclusively by the Read Greater operation. It may be shared by various portions of the program. The length of the key is the length defined for the key when the file was created, or is the partial key length specified for the operation.

A Read Greater operation may return >B4 or >B8 as the informative code. When the lock/unlock flag is set to one, the operation may also return >B7 as the informative code.

The following is an example of the source code to read a file record having a key greater than the specified key. The supervisor call block, read buffer, currency block, and key block are as follows:

```
RGK      DATA 0                        READ A RECORD OF FILE ASSIGNED
         BYTE >41,>5E                  TO LUNO >5E
         BYTE 0,>40
         DATA RGKRB
         DATA 80
         DATA 0
         DATA CRGK
RGKRB    BSS 80                        READ BUFFER
*---------------------------------- *  CURRENCY BLOCK
CRGK     BYTE 3                        PARTIAL KEY
         BYTE 3                        KEY NUMBER 3
         DATA RGKEY
CINRGK   BSS 16
RGKEY    TEXT 'TEX'
```

7.5.2.4 Read Greater or Equal. Sub-opcode >44 specifies a Read Greater or Equal operation. The Read Greater or Equal operation reads the first record of a key indexed file that contains a value equal to or greater than the specified value for a specified key. The operation stores the data in the buffer at the specified address. The characters are packed two per word. When the file contains more than one record having the requested key value or the immediately greater value, the operation reads the record that was inserted in the file first.

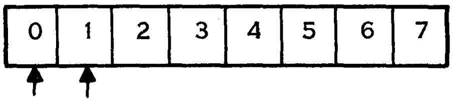
The following fields of the supervisor call block as extended for key indexed files apply to a Read Greater or Equal operation:

- SVC code — 0
- Return code
- Sub-opcode — >44
- Logical unit number (LUNO)
- System flags
- User flags
- Data buffer address
- Read character count
- <Actual read count>
- Currency block address

The following fields of the currency block apply to a Read Greater or Equal operation:

- <Informative code>
- Length of partial key
- Key number
- Key address

The following system flags apply to a Read Greater or Equal operation:

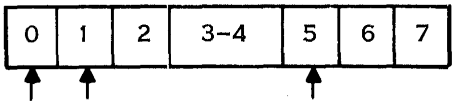


2279655

Bit 0 — Busy flag. Set by system as follows:
1 — Busy.
0 — Operation completed.

Bit 1 — Error flag. Set by system as follows:
1 — Error.
0 — No error.

The following user flags apply to a Read Greater or Equal operation:



2279656

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

Bit 1 — Key specified flag. Set to one for a Read Greater or Equal operation.

Bit 5 — Lock/unlock flag. Set as follows:
1 — File management locks the record after reading.
0 — Record remains in the state it was in before the read.

The LUNO field contains the LUNO assigned to the file.

The data buffer address contains the address of the buffer into which the record is to be read. The address must be an even number.

The read character count contains the maximum number of characters to be read into the buffer.

The actual read count is the number of characters read, returned by file management. The length returned is the length of the record or the length of the buffer, whichever is less. When an odd number of characters is read, an additional character is stored in the buffer, but the odd number (the actual number read) is placed in the actual read count field.

The length of the partial key field contains the length of the partial key for operations to which a partial key applies. The field must be set to zero or the defined length when the defined length of the key applies.

The key number is the number of the key as defined when the file was created. The MKF command displays the numbers of the keys of a key indexed file.

The key address is the address of the buffer that contains the key of the requested record. The key need not be in a buffer used exclusively by the Read Greater or Equal operation. It may be shared by various portions of the program. The length of the key is the length defined when the file was created, or the length of a partial key for a partial key operation.

A Read Greater or Equal operation may return >B4 or >B8 as the informative code. When the lock/unlock flag is set to one, the operation may also return >B7 as the informative code.

The following is an example of the source code to read a file record having a key equal to or greater than the specified key. The supervisor call block, read buffer, currency block, and key block are as follows:

```
RGEK      DATA 0              READ A RECORD OF FILE ASSIGNED
          BYTE >44,>5E        TO LUNO >5E
          BYTE 0,>40
          DATA RGEKRB
          DATA 80
          DATA 0
          DATA CRGEK
RGEKRB    BSS 80              READ BUFFER
CRGEK     BYTE 0              CURRENCY BLOCK
          BYTE 2              KEY NUMBER 2
          DATA RGEKEY
CNRGEK    BSS 16              ASSUMES KEY LENGTH IS 2
RGEKEY    TEXT 'TX'
```

7.5.2.5 Set Currency Equal. Sub-opcode >50 specifies a Set Currency Equal operation. The Set Currency Equal operation returns currency information for a record of a key indexed file that contains the specified key. Normally, one of the key indexed file operations that use currency information follows this operation. When the file contains more than one record having the requested key, the operation returns information for the record that was inserted in the file first.

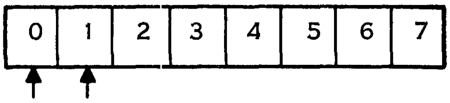
The following fields of the supervisor call block as extended for key indexed files apply to a Set Currency Equal operation:

- SVC code — 0
- Return code
- Sub-opcode — >50
- Logical unit number (LUNO)
- System flags
- User flags
- Currency block address

The following fields of the currency block apply to a Set Currency Equal operation:

- <Informative code>
- Length of partial key
- Key number
- Key address

The following system flags apply to a Set Currency Equal operation:

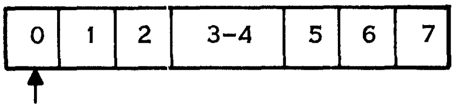


2279657

Bit 0 — Busy flag. Set by system as follows:
1 — Busy.
0 — Operation completed.

Bit 1 — Error flag. Set by system as follows:
1 — Error.
0 — No error.

The following user flags apply to a Set Currency Equal operation:



2279658

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

The LUNO field contains the LUNO assigned to the file.

The length of the partial key field contains the length of the partial key for operations to which a partial key applies. The equality of the partial key to the corresponding portion of the key in the record determines the selection of a record. The field must be set to zero or the defined length when the defined length of the key applies.

The key number is the number of the key as defined when the file was created. The MKF command displays the numbers of the keys of a key indexed file.

The key address is the address of the buffer that contains the key of the requested record. The key need not be in a buffer used exclusively by the Set Currency Equal operation. It may be shared by various portions of the program. The length of the key is the length defined when the file was created, or the length of a partial key for a partial key operation.

A Set Currency Equal operation may return >B4 or >B8 as the informative code.

The following is an example of the source code to obtain currency information for a file record having a key equal to the specified key. The supervisor call block, currency block, and key block are as follows:

```
SCEK      DATA 0               SET CURRENCY FOR A RECORD OF
          BYTE >50,>5E         FILE ASSIGNED TO
          BYTE 0,0             LUNO >5E
          DATA 0
          DATA 0
          DATA 0
          DATA SCEBK
SCEBK     BYTE 0               CURRENCY BLOCK
          BYTE 3               KEY NUMBER 3
          DATA SCEKEY
CEIN      BSS  16              ASSUMES KEY LENGTH IS 6
SCEKEY    TEXT 'AUSTIN'
```

7.5.2.6 Set Currency Greater. Sub-opcode >52 specifies a Set Currency Greater operation. The Set Currency Greater operation returns currency information for the first record of a key indexed file that contains a value greater than the specified value for a specified key. Normally, one of the key indexed file operations that use currency information follows this operation. When the file contains more than one record having the first key value greater than the requested key value, the operation returns information for the record that was inserted in the file first.

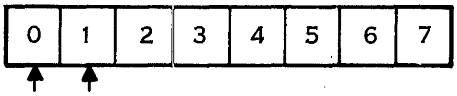
The following fields of the supervisor call block as extended for key indexed files apply to a Set Currency Greater operation:

- SVC code — 0
- Return code
- Sub-opcode — >52
- Logical unit number (LUNO)
- System flags
- User flags
- Currency block address

The following fields of the currency block apply to a Set Currency Greater operation:

- <Informative code>
- Length of partial key
- Key number
- Key address

The following system flags apply to a Set Currency Greater operation:

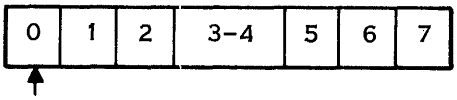


2279659

Bit 0 — Busy flag. Set by system as follows:
1 — Busy.
0 — Operation completed.

Bit 1 — Error flag. Set by system as follows:
1 — Error.
0 — No error.

The following user flags apply to a Set Currency Greater operation:



2279660

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

The LUNO field contains the LUNO assigned to the file.

The length of the partial key field contains the length of the partial key for operations to which a partial key applies. The field must be set to zero or the defined length when the defined length of the key applies.

The key number is the number of the key as defined when the file was created. The MKF command displays the numbers of the keys of a key indexed file.

The key address is the address of the buffer that contains the key of the requested record. The key need not be in a buffer used exclusively by the Set Currency Greater operation. It may be shared by various portions of the program. The length of the key is the length defined when the file was created, or the length of a partial key for a partial key operation.

A Set Currency Greater operation may return >B4 or >B8 as the informative code.

The LUNO field contains the LUNO assigned to the file.

The following is an example of the source code to obtain currency information for a file record having a key greater than the specified key. The supervisor call block, currency block, and key block are as follows:

```
SCGK     DATA  0              SET CURRENCY FOR A RECORD OF
         BYTE  >52,>5E        FILE ASSIGNED TO
         BYTE  0,0            LUNO >5E
         DATA  0
         DATA  0
         DATA  0
         DATA  CGBK
CGBK     BYTE  2              CURRENCY BLOCK
         BYTE  2              KEY NUMBER 2
         DATA  CGKEY
CGIN     BSS   16             PARTIAL KEY
CGKEY    TEXT  'AL'
```

7.5.2.7 Set Currency Greater or Equal. Sub-opcode >51 specifies a Set Currency Greater or Equal operation. The Set Currency Greater or Equal operation returns currency information for the first record of a key indexed file that contains a value equal to or greater than the specified value for a specified key. Normally, one of the key indexed file operations that use currency information follows this operation. When the file contains more than one record having the key equal to or greater than the requested key, the operation returns information for the record that was inserted in the file first.

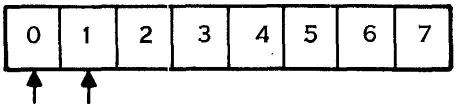
The following fields of the supervisor call block as extended for key indexed files apply to a Set Currency Greater or Equal operation:

- SVC code — 0
- Return code
- Sub-opcode — >51
- Logical unit number (LUNO)
- System flags
- User flags
- Currency block address

The following fields of the currency block apply to a Set Currency Greater or Equal operation:

- <Informative code>
- Length of partial key
- Key number
- Key address

The following system flags apply to a Set Currency Greater or Equal operation:

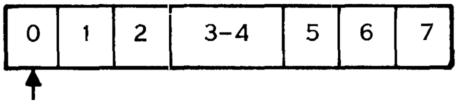


2279661

Bit 0 — Busy flag. Set by system as follows:
1 — Busy.
0 — Operation completed.

Bit 1 — Error flag. Set by system as follows:
1 — Error.
0 — No error.

The following user flags apply to a Set Currency Greater or Equal operation:



2279662

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

The LUNO field contains the LUNO assigned to the file.

The length of partial key field contains the length of the partial key for operations to which a partial key applies. The field must be set to zero or the defined length when the defined length of the key applies.

The key number is the number of the key as defined when the file was created. The MKF command displays the numbers of the keys of a key indexed file.

The key address is the address of the buffer that contains the key of the requested record. The key need not be in a buffer used exclusively by the Set Currency Greater or Equal operation. It may be shared by various portions of the program. The length of the key is the length defined when the file was created, or the length of a partial key for a partial key operation.

A Set Currency Greater or Equal operation may return >B4 or >B8 as the informative code.

The following is an example of the source code to obtain currency information for a file record having a key greater than or equal to the specified key. The supervisor call block, currency block, and key block are as follows:

```
SCGEK    DATA 0               SET CURRENCY FOR A RECORD OF
         BYTE >51,>5E         FILE ASSIGNED TO
         BYTE 0,0             LUNO >5E
         DATA 0
         DATA 0
         DATA 0
         DATA CGEK
CGEK     BYTE 0               CURRENCY BLOCK
         BYTE 3               KEY NUMBER 3
         DATA CGEKEY
CINGE    BSS  16              ASSUMES KEY LENGTH IS 3
CGEKEY   TEXT 'USA'
```

7.5.2.8 Delete by Key. Sub-opcode >49 specifies a Delete by Key operation. The Delete by Key operation deletes the record with a specified key. When the file contains more than one record having the specified key, the operation returns an informative code of >B4 and does not delete a record. A Delete Current operation (described in a subsequent paragraph) must be used to delete records having duplicate keys. A record that has been locked by a Read operation with the lock/unlock user flag set may not be deleted.

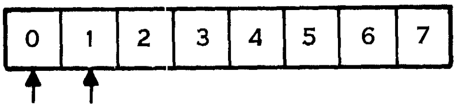
The following fields of the supervisor call block as extended for key indexed files apply to a Delete by Key operation:

- SVC code — 0
- Return code
- Sub-opcode — >49
- Logical unit number (LUNO)
- System flags
- User flags
- Currency block address

The following fields of the currency block apply to a Delete by Key operation:

- <Informative code>
- Key number
- Key address

The following system flags apply to a Delete by Key operation:

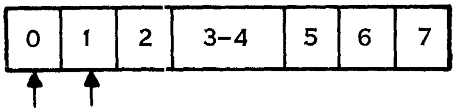


2279663

Bit 0 — Busy flag. Set by system as follows:
1 — Busy.
0 — Operation completed.

Bit 1 — Error flag. Set by system as follows:
1 — Error.
0 — No error.

The following user flags apply to a Delete by Key operation:



2279664

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

Bit 1 — Key specified flag. Set to one for a Delete by Key operation. The same sub-opcode with this flag set to 0 is a Delete Current operation.

The LUNO field contains the LUNO assigned to the file.

The key number is the number of the key as defined when the file was created. The MKF command displays the numbers of the keys of a key indexed file.

The key address is the address of the buffer that contains the key of the requested record. The key need not be in a buffer used exclusively by the Delete by Key operation. It may be shared by various portions of the program. The length of the key is the length defined when the file was created.

A Delete by Key operation may return >B4, >B5, or >B7 as the informative code.

The following is an example of the source code to delete a file record having a specified key. The supervisor call block, currency block, and key block are as follows:

```
DBK       DATA 0              DELETE A RECORD OF FILE
          BYTE >49,>5E        ASSIGNED TO LUNO >5E
          BYTE 0,>40
          DATA 0
          DATA 0
          DATA 0
          DATA DBKBK
DBKBK     BYTE 0              CURRENCY BLOCK
          BYTE 1              KEY NUMBER 1
          DATA DKEY
CINDK     BSS 16              ASSUMES KEY LENGTH IS 9
DKEY      TEXT 'LAZY LANE'
```

7.5.2.9 Read Current. Sub-opcode >42 specifies a Read Current operation. The Read Current operation reads a record of a key indexed file defined in currency information returned by a previous operation. The operation stores the data in the buffer at the specified address. The characters are packed two per word.

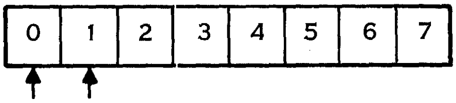
The following fields of the supervisor call block as extended for key indexed files apply to a Read Current operation:

- SVC code — 0
- Return code
- Sub-opcode — >42
- Logical unit number (LUNO)
- System flags
- User flags
- Data buffer address
- Read character count
- <Actual read count>
- Currency block address

The following fields of the currency block apply to a Read Current operation:

- <Informative code>
- Currency information

The following system flags apply to a Read Current operation:

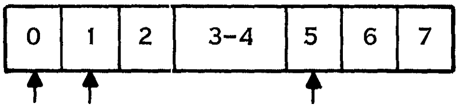


2279665

Bit 0 — Busy flag. Set by system as follows:
1 — Busy.
0 — Operation completed.

Bit 1 — Error flag. Set by system as follows:
1 — Error.
0 — No error.

The following user flags apply to a Read Current operation:



2279666

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

Bit 1 — Key specified flag. Set to 0 for a Read Current operation. The same sub-opcode is a Read by Key operation when this flag is set to one.

Bit 5 — Lock/unlock flag. Set as follows:
1 — File management locks the record after reading.
0 — Record remains in the state it was in before the read.

The LUNO field contains the LUNO assigned to the file.

The data buffer address contains the address of the buffer into which the record is to be read. The address must be an even number.

The read character count contains the maximum number of characters to be read into the buffer.

The actual read count is the number of characters read, returned by file management. The length returned is the length of the record or the length of the buffer, whichever is less. When an odd number of characters is read, an additional character is stored in the buffer, but the odd number (the actual number read) is placed in the actual read count field.

The currency information is in the currency block of a previous operation. Typically the same currency block is used for the Read Current operation.

A Read Current operation may return >B0 or >B3 as the informative code. When the lock/unlock flag is set to one, the operation may also return >B7 as the informative code.

The following is an example of the source code for the supervisor call block to read the current file record and for the read buffer. The currency block address is that of the currency block in the Set Currency Equal example.

```
RCK      DATA 0                    READ A RECORD OF FILE ASSIGNED
         BYTE >42,>5E              TO LUNO >5E
         BYTE 0,0
         DATA RCKRB
         DATA 80
         DATA 0
         DATA SCEBK
RCKRB    BSS 80                    READ BUFFER
```

7.5.2.10 Read Previous. Sub-opcode >48 specifies a Read Previous operation. The Read Previous operation reads the record of a key indexed file that precedes the record defined in currency information returned by a previous operation, other than a Set Currency operation. When the previous operation was a Set Currency operation, the record defined in the currency information is read. The operation stores the data in the buffer at the specified address. The characters are packed two per word.

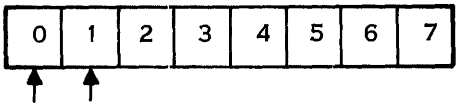
The following fields of the supervisor call block as extended for key indexed files apply to a Read Previous operation:

- SVC code — 0
- Return code
- Sub-opcode — >48
- Logical unit number (LUNO)
- System flags
- User flags
- Data buffer address
- Read character count
- <Actual read count>
- Currency block address

The following fields of the currency block apply to a Read Previous operation:

- <Informative code>
- Currency information

The following system flags apply to a Read Previous operation:

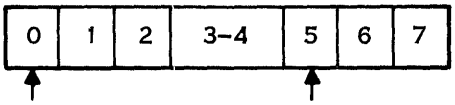


2279667

Bit 0 — Busy flag. Set by system as follows:
1 — Busy.
0 — Operation completed.

Bit 1 — Error flag. Set by system as follows:
1 — Error.
0 — No error.

The following user flags apply to a Read Previous operation:



2279668

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

Bit 5 — Lock/unlock flag. Set as follows:
1 — File management locks the record after reading.
0 — Record remains in the state it was in before the read.

The LUNO field contains the LUNO assigned to the file.

The data buffer address contains the address of the buffer into which the record is to be read. The address must be an even number.

The read character count contains the maximum number of characters to be read into the buffer.

The actual read count is the number of characters read, returned by file management. The length returned is the length of the record or the length of the buffer, whichever is less. When an odd number of characters is read, an additional character is stored in the buffer, but the odd number (the actual number read) is placed in the actual read count field.

The currency information is in the currency block of a previous operation. Typically the same currency block is used for the Read Previous operation.

A Read Previous operation may return >B3 or >BD as the informative code. When the lock/unlock flag is set to one, the operation may also return >B7 as the informative code.

The following is an example of the source code for the supervisor call block to read the previous record, and for the read buffer. The currency block address is that of the currency block in the Read Greater example.

```
RPK       DATA 0                    READ A RECORD OF FILE ASSIGNED
          BYTE >48,>5E              TO LUNO >5E
          BYTE 0,0
          DATA RPKRB
          DATA 80
          DATA 0
          DATA CRGK
RPKRB     BSS 80                    READ BUFFER
```

7.5.2.11 Read Next. Sub-opcode >45 specifies a Read Next operation. The Read Next operation reads the record of a key indexed file that follows the record defined in currency information returned by a previous operation, other than a Set Currency operation. When the previous operation was a Set Currency operation, the record defined in the currency information is read. The operation stores the data in the buffer at the specified address. The characters are packed two per word.

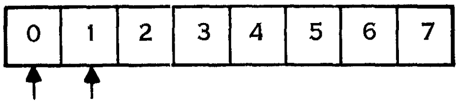
The following fields of the supervisor call block as extended for key indexed files apply to a Read Next operation:

- SVC code — 0
- Return code
- Sub-opcode — >45
- Logical unit number (LUNO)
- System flags
- User flags
- Data buffer address
- Read character count
- <Actual read count>
- Currency block address

The following fields of the currency block apply to a Read Next operation:

- <Informative code>
- Currency information

The following system flags apply to a Read Next operation:

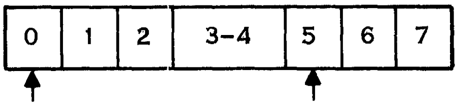


2279669

Bit 0 — Busy flag. Set by system as follows:
1 — Busy.
0 — Operation completed.

Bit 1 — Error flag. Set by system as follows:
1 — Error.
0 — No error.

The following user flags apply to a Read Next operation:



2279670

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

Bit 5 — Lock/unlock flag. Set as follows:
1 — File management locks the record after reading.
0 — Record remains in the state it was in before the read.

The LUNO field contains the LUNO assigned to the file.

The data buffer address contains the address of the buffer into which the record is to be read. The address must be an even number.

The read character count contains the maximum number of characters to be read into the buffer.

The actual read count is the number of characters read, returned by file management. The length returned is the length of the record or the length of the buffer, whichever is less. When an odd number of characters is read, an additional character is stored in the buffer, but the odd number (the actual number read) is placed in the actual read count field.

The currency information is in the currency block of a previous operation. Typically the same currency block is used for the Read Next operation.

A Read Next operation may return >B3 or >BD as the informative code. When the lock/unlock flag is set to one, the operation may also return >B7 as the informative code.

The following is an example of the source code for the supervisor call block to read the next record, and for the read buffer. The currency block address is that of the currency block in the Read Greater example.

```
RNK       DATA 0                        READ A RECORD OF FILE ASSIGNED
          BYTE >45,>5E                  TO LUNO >5E
          BYTE 0,0
          DATA RNKRB
          DATA 80
          DATA 0
          DATA CRGK
RNKRB     BSS 80                        READ BUFFER
```

7.5.2.12 Rewrite. Sub-opcode >47 specifies a Rewrite operation. The Rewrite operation replaces a record, with the following conditions:

- The operation must use the currency block set up by a Read operation.
- The previous Read operation must have locked the record.
- The new record need not be the same size.
- Only modifiable keys and data that are not included in any key may be altered.

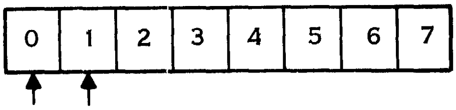
The following fields of the supervisor call block as extended for key indexed files apply to a Rewrite operation:

- SVC code — 0
- Return code
- Sub-opcode — >47
- Logical unit number (LUNO)
- System flags
- User flags
- Data buffer address
- Write character count
- Currency block address

The following fields of the currency block apply to a Rewrite operation:

- <Informative code>
- Currency information

The following system flags apply to a Rewrite operation:

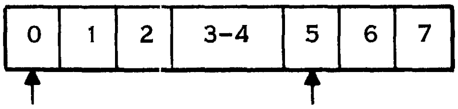


2279671

Bit 0 — Busy flag. Set by system as follows:
1 — Busy.
0 — Operation completed.

Bit 1 — Error flag. Set by system as follows:
1 — Error.
0 — No error.

The following user flags apply to a Rewrite operation:



2279672

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

Bit 5 — Lock/unlock flag. Set as follows:
1 — File management rewrites and unlocks the record.
0 — File management rewrites the record and it remains locked.

The LUNO field contains the LUNO assigned to the file.

The data buffer address is the address of the buffer that contains the record to be written. The address must be an even number.

The write character count is the number of characters to be written.

The currency information is in the currency block of a previous Read operation. The same currency block must be used for the Rewrite operation.

A Rewrite operation may return >BA as the informative code.

The following is an example of the source code for a supervisor call block to rewrite a record. The operation uses the currency block of the Read Greater example.

```
RWTKIF    DATA 0                  REWRITE RECORD OF FILE ASSIGNED
          BYTE >47,>5E            TO LUNO >5E
          BYTE 0,0
          DATA SFWB
          DATA 0
          DATA 80
          DATA CRGK
```

7.5.2.13 Unlock. Sub-opcode >4A specifies an Unlock operation. The Unlock operation releases exclusive control of any previously locked record, a record locked by another task or a record locked by the calling task. The record to be unlocked is identified by the currency information from a previous operation.

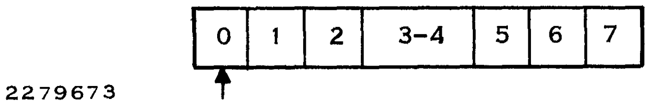
The following fields of the supervisor call block as extended for key indexed files apply to an Unlock operation:

- SVC code — 0
- Return code
- Sub-opcode — >4A
- Logical unit number (LUNO)
- User flags
- Currency block address

The following field of the currency block applies to an Unlock operation:

- Currency information

The following user flags apply to an Unlock operation:



Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

The LUNO field contains the LUNO assigned to the file.

The currency information is in the currency block of a previous operation. The same currency block may be used for the Unlock operation.

The following is an example of the source code for a supervisor call block to unlock a file record. The currency block referenced is the currency block for the Read Greater example.

```
UKIF     DATA 0                        UNLOCK FILE ASSIGNED TO LUNO >5E
         BYTE >4A,>5E
         BYTE 0,0
         DATA 0
         DATA 0
         DATA 0
         DATA CRGK
```

7.5.2.14 Delete Current. Sub-opcode >49 specifies a Delete Current operation. The Delete Current operation deletes the record specified by the currency information from a previous operation. A record that has been locked by a Read operation with the lock/unlock user flag set may not be deleted.

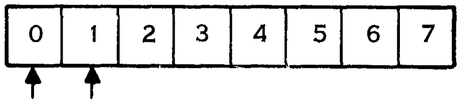
The following fields of the supervisor call block as extended for key indexed files apply to a Delete Current operation:

- SVC code — 0
- Return code
- Sub-opcode — >49
- Logical unit number (LUNO)
- System flags
- User flags
- Currency block address

The following fields of the currency block apply to a Delete Current operation:

- <Informative code>
- Currency information

The following system flags apply to a Delete Current operation:

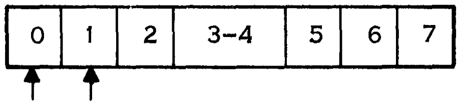


2279674

Bit 0 — Busy flag. Set by system as follows:
1 — Busy.
0 — Operation completed.

Bit 1 — Error flag. Set by system as follows:
1 — Error.
0 — No error.

The following user flags apply to a Delete Current operation:



2279675

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

Bit 1 — Key specified flag. Set to 0 for a Delete Current operation. The same sub-opcode with this flag set to one is a Delete by Key operation.

The LUNO field contains the LUNO assigned to the file.

The currency information is in the currency block of a previous operation. The same currency block may be used for the Delete Current operation.

A Delete Current operation may return >B4, >B5, >B7, or >BD as the informative code.

The following is an example of the source code for the supervisor call block to delete the current file record. The currency block address is that of the currency block in the Read Greater example.

```
DCR     DATA 0              DELETE A RECORD OF FILE
        BYTE >49,>5E        ASSIGNED TO LUNO >5E
        BYTE 0,0
        DATA 0
        DATA 0
        DATA 0
        DATA CRGK
```

7.5.2.15 Insert. Sub-opcode >46 specifies an Insert operation. The Insert operation writes a new record, making it available under every key defined for the file. The file may be thought of as if it were a set of sequential files, each corresponding to a key and sorted on its key. The effect of an Insert operation is to insert the record in the proper position in each of these simulated files according to the key.

The primary key of the inserted record can be a null key (all blanks or having >FF in the first byte); the record can be accessed by this key. Any secondary key that can be modified can be a null key, but the key is not placed in the file; therefore the record cannot be accessed by this key.

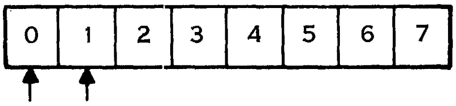
The following fields of the supervisor call block as extended for key indexed files apply to an Insert operation:

- SVC code — 0
- Return code
- Sub-opcode — >46
- Logical unit number (LUNO)
- System flags
- User flags
- Data buffer address
- Write character count
- Currency block address

The following fields of the currency block apply to an Insert operation:

- <Informative code>
- Currency information

The following system flags apply to an Insert operation:

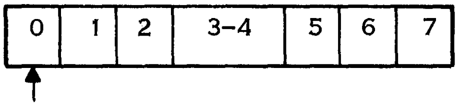


2279676

Bit 0 — Busy flag. Set by system as follows:
1 — Busy.
0 — Operation completed.

Bit 1 — Error flag. Set by system as follows:
1 — Error.
0 — No error.

The following user flags apply to an Insert operation:



2279677

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

The LUNO field contains the LUNO assigned to the file.

The data buffer address is the address of the buffer that contains the record to be written. The address must be an even number.

The write character count is the number of characters to be written.

The currency information that defines the inserted record is returned in the currency block.

An Insert operation may return >B4 as the informative code.

The following is an example of the source code for a supervisor call block to insert a record, and for the currency block:

```
IKIF     DATA 0                     INSERT RECORD IN FILE
         BYTE >46,>5E               ASSIGNED TO LUNO >5E
         BYTE 0,0
         DATA SFWB
         DATA 0
         DATA 80
         DATA CIK
CIK      BYTE 0,0                   CURRENCY BLOCK
         DATA 0
CINIK    BSS 16
SFWB     TEXT 'JOHN CUE PUBLIC'     THE RECORD TO BE INSERTED
```

8

Interprocess Communication

8.1 INTRODUCTION

Communication between programs, or interprocess communication (IPC), allows programs to exchange data. IPC operations are pseudo-I/O operations because each of the programs involved in the exchange acts as a peripheral device or file with respect to the other.

This section describes interprocess communication (IPC) concepts, utility operations required for IPC, and IPC I/O operations. IPC supports resource-independent I/O to symmetric channels and resource-specific I/O to master/slave channels. Descriptions of IPC I/O operations are arranged according to the type of channel to which they apply.

8.2 COMMUNICATING BETWEEN TASKS

DNOS provides a means of communicating between tasks called interprocess communication (IPC). A communication path between two or more tasks is called a channel. I/O operations are directed to channels to accomplish interprocess communication.

Utility operations create and delete IPC channels. The Assign LUNO utility operation assigns a LUNO to a channel. I/O operations open and close the channel, and perform the actual transfers of data to provide interprocess communication.

Resource-independent I/O is performed to symmetric IPC channels. A symmetric IPC channel provides transfer of messages between the owner task and a requesting task. The matching of a request for an I/O operation to a symmetric channel by one task with a request from another task provides a task synchronization function; the message transferred may be incidental to the synchronization function. The message may be a short procedural message or a transfer of data between tasks.

Transfer of messages in a symmetric channel is either to or from the owner task. A write request of one task must be matched by a read request from another task. The requesting task may be any task, as restricted by the scope of the channel (specified when the channel is created) and the access privileges of the LUNO assigned to the channel (specified when the owner task opened the LUNO). A message issued by the owner task is received by the task that first issues a read request. A Read operation initiated by the owner task receives the message in the first write request directed to the channel.

Typical applications of symmetric channels include cases of owner tasks receiving messages from several requesting tasks. In this case, the owner task issues a read request. When a requesting task issues a write request, the message is transferred. The owner task performs the function appropriate for the message and issues another read request. When a requesting task has issued another write request during the processing of the first request, the second message is transferred when the owner task completes the first transfer and issues another read request. Otherwise, the owner task may be suspended until a requesting task issues a write request to the channel. If several requesting tasks issue a write request, each request is placed into a queue and is transferred when the owner task issues succeeding read requests. The owner task is not suspended if several requesting tasks issue a write request to the channel.

The owner task of a symmetric channel may send messages to several requesting tasks. The owner task prepares (or updates) the message, and issues a write request. The message is transferred when a requesting task issues a read request. When the transfer is complete, the owner task performs any required processing and issues another write request. When a read request from a requesting task is pending, the message is immediately transferred. Otherwise, the owner task may be suspended until a requesting task issues a read request to the channel. These single directional tasks may have an unlimited number of requesting tasks.

A symmetric channel may perform a bidirectional transfer of messages with a requesting task. The owner task may prepare a message and issue a write request. If there is a read request from the requesting task pending, the message is transferred immediately. Otherwise, the transfer occurs when the requesting task issues a read request. Next, each task performs appropriate processing. Then the requesting task issues a write request, and the owner task issues a read request (or the owner task may issue a read request before the requesting task issues the write request). When both requests have been issued, the reply message is transferred. The tasks can continue to communicate in this way until no further communication is required. In another type of bidirectional exchange of messages, the first message exchanged is from the requesting task to the owner task; the exchange continues in a similar manner. This type of bidirectional channel may have only one requesting task.

Resource-specific I/O is performed to master/slave IPC channels. A default resource is specified for each master/slave channel when it is created. For example, when the default resource for a channel is the 911 VDT, any operation that could be directed to a 911 VDT may be directed to the channel. The channel becomes a surrogate VDT to requesting tasks. The owner task of a master/slave channel performs the functions of the default resource. It obtains the requests for the channel and processes them appropriately. The owner task can be thought of as a software device or file that performs functions that may be requested of a device or file. The default resource for a channel may be a channel. In this case, any resource-independent request may be directed to the channel.

The owner task of a master/slave IPC channel may process sub-opcodes for the default resource in a special way. In that case, the programmer of the owner task must adequately document the manner in which the task responds to each sub-opcode. The capabilities of master/slave channel owner tasks are very flexible; they must be used properly to be useful.

Scope considerations apply to master/slave channels in the same way as they apply to symmetric channels. However, the owner task of a master/slave channel may process Assign LUNO requests (and Release LUNO requests) in addition to other I/O requests. In this way the owner task can identify the task with which it is communicating and more effectively control the communication.

The basic supervisor call block shown in Section 5 applies to IPC operations. Any variations required are noted in the descriptions of specific operations. The flags used for each operation are shown in the description of the operation. The basic supervisor call block is as follows:

SVC > 00 -- I/O Operations

Align on word boundary
Can be initiated as an event

Dec	Hex		
0	0	>00	<Return Code>
2	2	Sub-Opcode	LUNO
4	4	<System Flags>	User Flags
6	6	Data Buffer Address	
8	8	Read Character Count	
10	A	Write Character Count/<Actual Read Count>	

2279678

8.3 IPC UTILITY OPERATIONS

IPC operations utilize channels in communicating between programs and require utility functions to create and delete channels, to assign LUNOs to channels, to release LUNOs, and to apply and remove protection to the channels.

8.3.1 Performing Utility Functions

A subset of the sub-opcodes of the I/O Operations SVC (opcode 00) performs I/O utility functions that support interprocess communication. These I/O utility functions allow a program to:

- Assign a LUNO
- Release a LUNO
- Apply write protection
- Apply delete protection
- Remove protection
- Create an IPC channel
- Delete an IPC channel

Many of these utility operations require pathnames. The pathname of an IPC channel is similar to that of the program file that contains the owner task. That is, both pathnames consist of the same volume name and directory names. The channel name replaces the final component of the program file pathname. The channel name is chosen when the channel is created.

Interprocess communication requires channels to which messages are written and from which messages are read. Utility functions create and delete IPC channels. An IPC channel is either a symmetric channel or a master/slave channel, and is under control of a user task in either case.

Channels are created without protection. A utility function of the I/O Operations SVC applies delete protection, and another function of the SVC removes protection.

A task must assign a LUNO to an IPC channel and open that LUNO before attempting to read or write to the channel.

The first task to use a symmetric channel issues a read or write request and is suspended pending a matching write or read request from another task (the requesting task). The functions of a symmetric channel are resource-independent.

The owner task of a master/slave channel exercises a greater degree of control over the channel than does the owner of a symmetric channel. The functions of a master/slave channel are resource-specific.

The owner task processes requests for channel operations. A channel is created with a default resource. Tasks request operations on the channel as if it were the default resource. The owner task issues a master read request and receives the information supplied in a request by another task. The owner task performs the requested operation, modifying the information in the same way as the default resource would modify it. The owner task then issues a master write request that returns the modified information to the requesting task.

The extended supervisor call blocks for creating and deleting an IPC channel are shown in the paragraphs that describe those operations. The other utility operations use the extended call block shown in the following:

SVC >00 -- I/O OPERATIONS
(UTILITY SUB-OPCODES)

ALIGN ON WORD BOUNDARY
CAN BE INITIATED AS AN EVENT

DEC	HEX		
0	0	>00	<RETURN CODE>
2	2	SUB-OPCODE	LUNO
4	4	<SYSTEM FLAGS>	USER FLAGS
6	6	<RESOURCE TYPE>	
8	8	RESERVED	
10	A		
12	C	KEY DEF. BLOCK ADDR/DEF. PHYS. REC. SIZE	
14	E	RESERVED	
16	10	UTILITY FLAGS	
18	12	DEFINED LOGICAL RECORD LENGTH	
20	14	DEFINED PHYSICAL RECORD LENGTH	
22	16	PATHNAME ADDRESS	
24	18	RESERVED	
26	1A		
28	1C	INITIAL FILE ALLOCATION	
30	1E		
32	20	SECONDARY FILE ALLOCATION	
34	22		

2279679

8.3.1.1 Creating an IPC Channel. An interprocess communication (IPC) channel is a communication path between two or more tasks, with one of the tasks designated as the owner. The owner task has control over how the channel is used. The owner task is designated when the channel is created.

A channel is created either as a symmetric channel or as a master/slave channel. A symmetric channel provides resource-independent I/O; either the owner task or a requesting task may request Read and Write operations. When the owner task issues a write request, the Write operation is deferred until a requesting task issues a read request. When a requesting task issues a write request, the Write operation is deferred until the owner task issues a read request. Note that the destination of a message sent by the owner task is the requesting task that first issues a read request. Also, the source of a message received by the owner task is the task that first issued a write request.

A master/slave channel provides resource-specific I/O; the owner task processes all I/O requests of requesting tasks. A default resource type is defined for the channel when it is created. Requesting tasks access the channel as if it were the default resource. The default resource type of a channel must be appropriate for the operations to be performed on the channel. For a Write with Reply operation directed to the channel to be successful, the default resource type must be that of a device for which the operation is valid; for file operations to be performed, the resource type must be sequential file or relative record file. Except for those cases in which the master task performs the functions of a device or file, channel should be specified as the default resource type. When channel is the default resource type, the function of the master/slave channel is primarily the transfer of messages, and no special sub-opcodes or flags are required.

The owner task performs a Master Read operation to obtain the oldest request directed to the channel. The owner task issues a Master Write request to return information to the requesting task. The information transferred includes a data structure derived from the supervisor call block for the request and appropriate buffers.

The owner task of a master/slave channel has the option of processing assign LUNO requests from other tasks. The option may be selected by setting a flag when the channel is created. When the option is selected, the I/O utility processor passes the assign LUNO request to the owner task of the channel after performing initial processing. The owner task returns the request to the I/O utility processor after processing by executing a master write request.

The owner task may or may not be replicatable, and may be bid automatically when a LUNO is assigned to the channel. The scope of the channel determines the replicatability of the owner task as it functions as the channel owner task. The scope of the channel also determines whether or not an Assign LUNO operation for the channel may bid the owner task. When an owner task is executed independently of its function as a channel owner task, the replicatability specified when the task was installed applies.

The scope of a channel is specified when a channel is created. The channel may have global, job-local, or task-local scope.

The characteristics of a channel having global scope are:

- Any task in the system may access the channel.
- The owner task is not replicated.
- The owner task must access the channel before any other task.
- An Assign LUNO operation does not bid the owner task.

The characteristics of a channel having job-local scope are:

- Only tasks in the same job as the owner task may access the channel.
- If the owner task owns only one channel, then the owner task is never replicated in the same job, but it may be replicated in other jobs. If the owner task owns more than one channel but is not using the channel specified by the requestor task, it may be replicated in the same job.
- Either the owner task or another task may access the channel first.
- An Assign LUNO operation may bid the owner task.

The characteristics of a channel having task-local scope are:

- Only one (requester) task may access the channel.
- The owner task is replicated for each requesting task.
- The requesting task must access the channel before the owner task.
- An Assign LUNO operation bids the owner task. The owner task may not be bid directly.

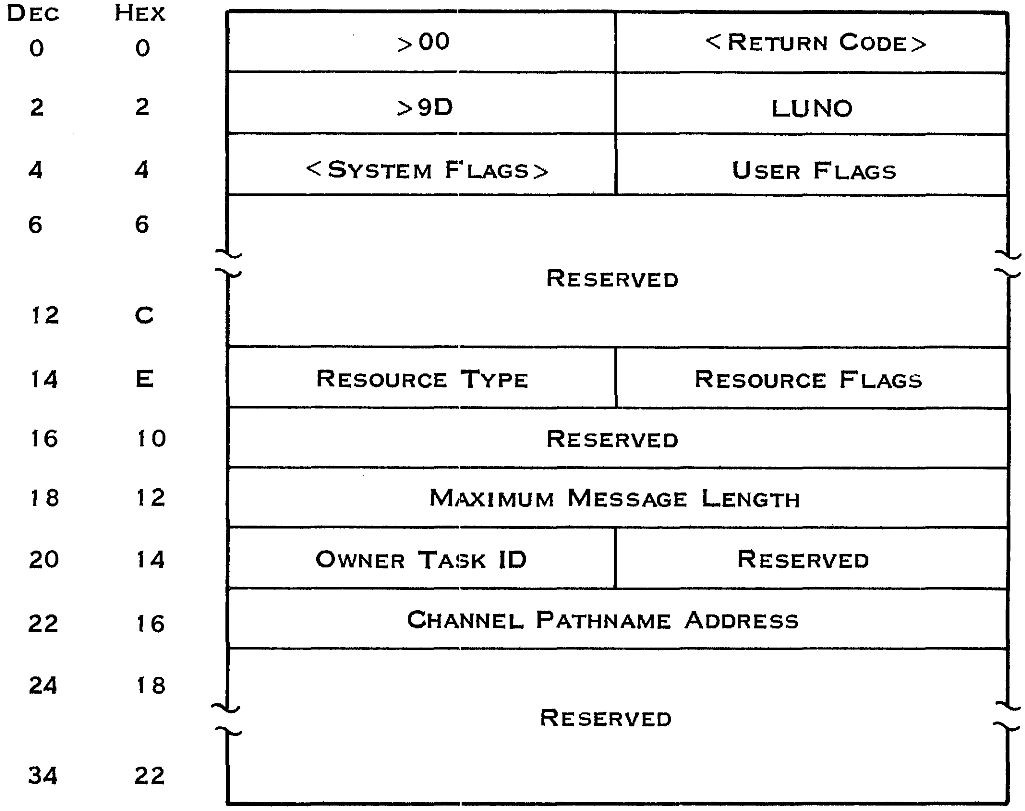
An owner task may be the owner of only one task-local channel at a time. An owner of a job-local or global channel may be the owner task of other channels having the same scope. A create channel request that would result in an owner task being the owner of channels with different scopes is not allowed.

An IPC channel may be created as a shared channel. The shared attribute of a channel is distinct from the access privileges specified in an Open operation. A channel that is not shared may be opened with any type of access privileges, but it is only available to one requesting task at a time. A channel that is shared is available to any number of requesting tasks, depending on the access privileges specified in the Open operations. A channel that has a task-local scope may not be shared, whether or not it is created with a shared attribute.

To create an IPC channel, a program executes an I/O operations SVC with sub-opcode >9D. The extended supervisor call block for this operation is as follows:

SVC >00 -- I/O OPERATIONS
(UTILITY SUB-OPCODE >9D)

ALIGN ON WORD BOUNDARY
CAN INITIATED AS AN EVENT



DEC	HEX		
0	0	>00	<RETURN CODE>
2	2	>9D	LUNO
4	4	<SYSTEM FLAGS>	USER FLAGS
6	6	RESERVED	
12	C		
14	E	RESOURCE TYPE	RESOURCE FLAGS
16	10	RESERVED	
18	12	MAXIMUM MESSAGE LENGTH	
20	14	OWNER TASK ID	RESERVED
22	16	CHANNEL PATHNAME ADDRESS	
24	18	RESERVED	
34	22		

2279680

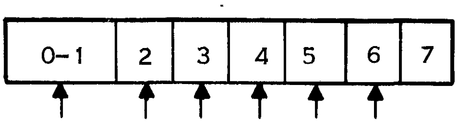
The call block contains the following:

Byte	Contents
0	Opcode, >00.
1	Return code. DNOS returns zero when the operation completes satisfactorily. When the operation completes in error, DNOS returns an error code.
2	Utility sub-opcode >9D.
3	LUNO assigned to owner task program file, or zero when owner task is on .S$SHARED, the shared program file.

Byte **Contents**

4 <System flags>.

5 User flags:



2279681

Bits 0–1 — Channel scope flag. Set as follows:
00 — Task-local scope.
01 — Job-local scope.
10 — Global scope.
Bit 2 — Shared channel flag. Set as follows:
1 — Shared channel.
0 — Not a shared channel.
Bit 3 — Channel type flag. Set as follows:
1 — Symmetric channel.
0 — Master/slave channel.
Bit 4 — Owner process assign flag. Set as follows:
1 — Owner task processes Assign LUNO operations. (Master/slave channel only.)
0 — Owner task does not process Assign LUNO operations.
Bit 5 — Owner process abort flag. Set as follows:
1 — Owner task processes Abort I/O operation (SVC >0F).(Master/slave channel only.)
0 — Owner task does not process Abort I/O operation.
Bit 6 — Owner process all utility opcodes. Set as follows:
1 — Owner task processes all I/O utility operations (SVC >00, subopcodes >90 and higher) including Assign LUNO. (Master/slave channel only.)
0 — Owner task does not process all I/O utility operations.
Bit 7 — Reserved.

6–13 Reserved.

14 Default resource type. Set to zero for a symmetric channel. For a master/slave channel, set as follows to specify default resource type:

When channel default resource flag is set in byte 15, bit 5:
0 — Resource-independent operations apply.

When device default resource flag is set in byte 15, bit 6:
0 — Dummy device.
1 — Special device.
2 — 743 KSR.
3 — 733 ASR.
4 — 733 cassette drive.

Byte	**Contents**
	6 — Single-sided diskette drive. 7 — Disk drive. 8 — Magnetic tape drive. 9 — Teleprinter device (TPD). 10 — 911 VDT. 11 — Serial printer. 12 — Parallel printer. 13 — Four-channel communication controller (FCCC). 14 — Communication interface module (CIM). 15 — Industrial device. 16 — Card reader. 17 — 940 VDT. 18 — 931 VDT 20 — Bit-oriented/character-oriented asynchronous interface module (BCAIM). 21 — Virtual Terminal. When file default resource flag is set in byte 15, bit 7: 0 — Reserved. 1 — Sequential file. 2 — Relative record file. 3 — Key indexed file. 4 — Directory file. 5 — Program file. 6 — Image file.
15	Default resource flags (only one may be set): Bits 0-3 — Reserved. Bit 4 — Remote flag. 1 — Opens are not validated. 0 — Opens are validated. Bit 5 — Channel resource flag. Set to one for a symmetric channel. For a master/slave channel, set as follows: 1 — Default resource is an IPC channel. 0 — Default resource is not a channel. Bit 6 — Device resource flag. Set as follows: 1 — Default resource is a device. 0 — Default resource is not a device. Bit 7 — File resource flag. Set as follows: 1 — Default resource is a file. 0 — Default resource is not a file.

Byte	Contents
16-17	Reserved.
18-19	Maximum message length. Maximum length of message transferred on this channel. The maximum size is 12,288 (>3000) characters.
20	Owner task ID. The installed ID of the owner task on its program file.
21	Reserved.
22-23	Channel pathname address. Address of a buffer that contains the channel pathname (see text that follows).
24-35	Reserved.

The channel pathname is identical to the owner task pathname except in the final component. The channel name is arbitrarily chosen and must be unique among the file names, directory names, aliases, and names of any other channels listed in the directory. An LD command may be entered to list the contents of the directory, including any channels in the directory. The following example shows a channel pathname that would be appropriate for an owner task in program file PROGS of directory VOL4.GREEN.COMM.

Owner Task Program File — VOL4.GREEN.COMM.PROGS

Valid Channel Name — VOL4.GREEN.COMM.CHAN1

The buffer contains the length of the pathname in the first byte, followed by the characters of the pathname in succeeding bytes.

The following is an example of the source code for a supervisor call block to create a channel:

```
CCHAN    DATA 0                    CREATE SYMMETRIC IPC CHANNEL,
         BYTE >9D                  PATHNAME VOL3.BLUE.CH2. OWNER TASK
         BYTE >1F                  INSTALLED ID IS >2A. PROGRAM FILE
         BYTE 0                    LUNO IS >1F. CHANNEL SCOPE IS TASK
         BYTE >10                  LOCAL. CHANNEL IS NOT SHARED.
         DATA 0,0,0,0              MESSAGES LIMITED TO 80
         BYTE 0                    CHARACTERS
         BYTE >04
         DATA 0
         DATA 80
         BYTE >2A
         BYTE 0
         DATA CHPTHN
         DATA 0,0,0,0,0,0
CHPTHN   BYTE CNEND-$
         TEXT ‘VOL3.BLUE.CH2’
CNEND    EQU  $-1
```

8.3.1.2 Deleting an IPC Channel. When a program file that contains channel owner tasks is deleted, the channels owned by the owner tasks are deleted also. Channels cannot be protected from being deleted in this manner. That is, the channel may be delete protected, but the delete protection of the channels is ignored if the program file is not also delete protected.

An IPC channel can be deleted independently of the program file. Delete protection that has been applied to the channel must be removed prior to deleting the channel.

To delete an IPC channel, a program executes an I/O operations SVC with sub-opcode >9E. The extended supervisor call block for this operation is as follows:

SVC >00 -- I/O Operation
(Utility sub-opcode>9E)

Align on word boundary
Can be initiated as an event

Dec	Hex		
0	0	>00	<Return Code>
2	2	>9E	Reserved
4	4	Reserved	
20	14		
22	16	Channel Pathname Address	
24	18	Reserved	
34	22		

2279682

The call block contains the following:

Byte	Contents
00	Opcode, >00.
1	Return code. DNOS returns zero when the operation completes satisfactorily. When the operation completes in error, DNOS returns an error code.
2	Utility sub-opcode >9E.
3-21	Reserved.
22-23	Channel pathname address. Address of a buffer that contains the channel pathname.
24-35	Reserved.

The buffer contains the length of the pathname in the first byte, followed by the characters of the pathname in succeeding bytes.

The following is an example of the source code for a supervisor call block to delete a channel:

```
DCHAN    DATA 0                      DELETE IPC CHANNEL, PATHNAME
         BYTE >9E                    VOL3.BLUE.CH2
         BYTE >0
         DATA 0,0,0,0,0
         DATA 0,0,0,0
         DATA CPTHN
         DATA 0,0,0,0,0,0
CPTHN    BYTE CEND-$
         TEXT 'VOL3.BLUE.CH2'
CEND     EQU  $-1
```

8.3.1.3 Assigning LUNOs. To assign a LUNO, a program executes an I/O Operations SVC with sub-opcode >91. The following fields of the utility supervisor call block apply:

- SVC code — 0
- Return code
- Utility sub-opcode — >91
- Logical unit number (LUNO)
- <Resource type>
- Utility flags
- Pathname address

The system returns the resource type in bytes 6 and 7 of the call block. Byte 6 indicates the device type and byte 7 indicates the resource type.

The device types found in byte 6 are as follows:

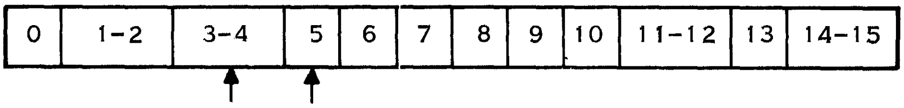
>00 — Dummy device
>01 — Special device
>02 — 743 KSR
>03 — 733 ASR
>04 — 733 cassette drive
>06 — Single-sided diskette drive
>07 — Disk drive
>08 — Magnetic tape drive
>09 — Teleprinter device (TPD)
>0A — 911 VDT
>0B — Serial printer
>0C — Parallel printer
>0D — Four-channel communication controller (FCCC)
>0E — Communication interface module (CIM)
>0F — Industrial device
>10 — Card reader
>11 — 940 VDT
>12 — 931 VDT
>14 — Bit-oriented/character-oriented asynchronous interface module (BCAIM)
>15 — Virtual terminal

The resource types found in byte 7 are as follows:

>01 — File
>02 — Device
>04 — Channel
>08 — Remote

The value in byte 7 is formed by DNOS by using one or more of the values listed above. Some of the values are combinations of these values. For example, >06 is a channel emulating a device. If the value in bytes 6 and 7 is >0A06, the indicated resource type is a channel emulating a 911 VDT. If the value in byte 7 indicates the remote bit, the LUNO is assigned to a resource on another system and no local checking is performed locally for open access conflicts.

The following utility flags apply:



2279683

Bits 3-4 — Scope of LUNO flag. Set as follows:
00 — Task-local LUNO.
01 — Job-local LUNO.
10 — Global LUNO.
11 — Job-local-shared LUNO.

Bit 5 — Generate LUNO flag. Set as follows:
1 — Assign the next available LUNO and return LUNO in byte 3.
0 — Assign the LUNO specified in byte 3.

Set all other utility flags to zero.

A logical unit number (LUNO) must be assigned to an I/O resource to identify the resource for an I/O operation. The scope of a global LUNO is not limited to a single job or task. The LUNO applies in all jobs and tasks executing while it remains assigned. The scope of a job-local LUNO is limited to the tasks in the job. A job-local LUNO is assigned by one of the tasks in the job or by an SCI command. The scope of a task-local LUNO is limited to the task that assigns the LUNO. A task-local LUNO is assigned by a task.

Job-local-shared LUNOs (shared LUNOs) are job-local LUNOs that can be used by more than one task within a given job. Once the LUNO is opened it can be used by other tasks within the same job. These other tasks need not open it. If they open it, the access privileges of the LUNO are compared to those requested in the Open operation. If the Open operation requests greater access privileges and it does not conflict with the access privileges of other LUNOs that are assigned and opened to the resource, the privilege level of the LUNO is changed to the greater value. The access privileges of a LUNO in order of increasing value are read only, shared, exclusive write, exclusive all. If the requested access privilege is less than or equal to the present value, the privilege level of the LUNO is not changed. Thus, all tasks that use a shared LUNO have the same access privileges to the resource regardless of how they opened it.

A count of the number of successful Open operations is kept. The same number of Close operations must be performed before the LUNO can be released. If a Close operation is not performed the LUNO is not released until the job terminates.

The use of shared LUNOs tends to reduce the total number of LUNOs required in the system. This type of LUNO is not recommended for sequential files because there is no defined method of positioning the file; that is, the task has no control of which record is read or written.

The Assign LUNO operation may assign the next available LUNO or a LUNO specified in the LUNO field. When the generate LUNO flag is set to one, the system assigns the next available LUNO and returns the number in the LUNO field. When the flag is set to zero, the system considers the contents of the LUNO field of the supervisor call block to be the desired LUNO.

The pathname address is the address of an area of memory that contains the pathname of a channel to which the LUNO is assigned. The first byte of the pathname area contains the number of characters in the pathname. Subsequent bytes contain the ASCII characters of the pathname.

The following is an example of the source code for a supervisor call block and the pathname block to assign a LUNO to a file:

```
ALUNO     DATA 0                          ASSIGN TASK LOCAL LUNO >18 TO
          BYTE >91                        CHANNEL VOL3.BILL.PROGS.CHAN1
          BYTE >18
          DATA 0,0
          DATA 0,0
          DATA 0,0
          BYTE 0,0                        UTILITY FLAGS
          DATA 0,0
          DATA PNME
          DATA 0,0
          DATA 0,0
          DATA 0,0
PNME      BYTE N4-$                       PATHNAME LENGTH
          TEXT 'VOL3.BILL.PROGS.CHAN1'
N4        EQU  $-1
```

Table 8-1 shows the types of LUNOs that can be assigned to IPC channels with different scopes.

Table 8-1. IPC Channel and LUNO Scope

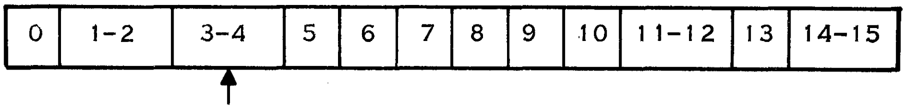
	LUNO Scope		
Channel Scope	Task	Job	Global
Global	x	x	x
Job-local	x	x	
Task-local	x	x	

2279760

8.3.1.4 Releasing LUNOs. To release a LUNO, a program executes an I/O Operations SVC with sub-opcode >93. The following fields of the utility supervisor call block apply:

- SVC code — 0
- Return code
- Utility sub-opcode — >93
- Logical unit number (LUNO)
- Utility flags

The following utility flags apply:



2279684

Bits 3-4 — Scope of LUNO. Set as follows:
00 — Task-local LUNO
01 — Job-local LUNO
10 — Global LUNO
11 — Job-local-shared LUNO

Set all other utility flags to zero.

A Release LUNO operation does not release a LUNO that has a different scope from that specified by the scope of LUNO flag. For example, if global LUNO >23, job-local LUNO >23, and task-local LUNO >23 were all assigned, and a Release LUNO operation for task-local LUNO >23 were performed, the global and job-local LUNOs would remain assigned.

The following is an example of the source code for a supervisor call block to release a LUNO:

```
RLUNO    DATA 0                    RELEASE GLOBAL LUNO >23.
         BYTE >93
         BYTE >23
         DATA 0,0
         DATA 0,0
         DATA 0,0
         BYTE >10,0                UTILITY FLAGS
         DATA 0,0
         DATA 0
         DATA 0,0
         DATA 0,0
         DATA 0,0
```

8.3.1.5 Write Protecting Channels. To write protect a channel, a program executes an I/O Operations SVC with sub-opcode >97. IPC channel write protection only prevents accidental deletion of the channel. The following fields of the utility supervisor call block apply:

- SVC code — 0
- Return code
- Utility sub-opcode — >97
- Pathname address

All utility flags should be set to zero.

Channels are created with no protection. When write protection is applied to an IPC channel, the channel is protected from being deleted. The protection afforded by write protection applies to the data structure that implements the channel, not to transfers of messages between tasks. Write protection must be removed from an IPC channel before it may be deleted by a Delete IPC Channel operation.

The pathname address is the address of an area of memory that contains the pathname of the channel to be write-protected. The byte at the pathname address contains the number of characters in the pathname. Subsequent bytes contain the ASCII characters of the pathname.

The following is an example of the source code for a supervisor call block to write protect a channel:

```
WRPR    DATA 0                  APPLY DELETE PROTECTION TO CHANNEL
        BYTE >97                VOL5.SOURCE.PROGA.CHAN
        BYTE 0
        DATA 0
        DATA 0
        DATA 0,0
        DATA 0,0
        BYTE 0,0                UTILITY FLAGS
        DATA 0,0
        DATA PNAM
        DATA 0,0
        DATA 0,0
        DATA 0,0
PNAM    BYTE PN1-$              PATHNAME LENGTH
        TEXT 'VOL5.SOURCE.PROGA.CHAN'
PN1     EQU  $-1
```

8.3.1.6 Delete Protecting Channels. To delete protect a channel, a program executes an I/O Operations SVC with sub-opcode >98. The following fields of the utility supervisor call block apply:

- SVC code — 0
- Return code
- Utility sub-opcode — >98
- Pathname address

All utility flags should be set to zero.

Files are created with no protection. After a Delete Protect operation is performed, a Delete IPC Channel operation may not be performed on the channel. Protection is removed by performing a Remove Protection operation.

The pathname address is the address of an area of memory that contains the pathname of the channel to be delete protected. The byte at the pathname address contains the number of characters in the pathname. Subsequent bytes contain the ASCII characters of the pathname.

The following is an example of the source code for a supervisor call block to delete protect a channel:

```
DPR      DATA 0                    APPLY DELETE PROTECTION TO CHANNEL
         BYTE >98                  VOL1.SOURCE.PROGB.CHAN
         BYTE 0
         DATA 0
         DATA 0
         DATA 0,0
         DATA 0,0
         BYTE 0,0                  UTILITY FLAGS
         DATA 0,0
         DATA DPNAM
         DATA 0,0
         DATA 0,0
         DATA 0,0
DPNAM    BYTE DPN1-$               PATHNAME LENGTH
         TEXT 'VOL1.SOURCE.PROGB.CHAN'
DPN1     EQU  $-1
```

8.3.1.7 Removing Channel Protection. To remove protection from an IPC channel, a program executes an I/O Operations SVC with sub-opcode >96. The following fields of the utility supervisor call block apply:

- SVC code — 0
- Return code
- Utility sub-opcode — >96
- Pathname address

All utility flags should be set to zero.

When write or delete protection has been applied to a channel, the channel remains protected until a Remove Protection operation is performed on the channel. The Remove Protection operation removes both write protection and delete protection, leaving the channel unprotected.

The pathname address is the address of an area of memory that contains the pathname of the channel from which protection is to be removed. The byte at the pathname address contains the number of characters in the pathname. Subsequent bytes contain the ASCII characters of the pathname.

The following is an example of the source code for a supervisor call block to remove protection from a channel:

```
RPROT    DATA 0                       REMOVE PROTECTION FROM CHANNEL
         BYTE >96                     USING PATHNAME BLOCK OF
         BYTE 0                       PRECEDING EXAMPLE
         DATA 0
         DATA 0
         DATA 0,0
         DATA 0,0
         BYTE 0,0                     UTILITY FLAGS
         DATA 0,0
         DATA DPNAM
         DATA 0,0
         DATA 0,0
         DATA 0,0
```

8.3.2 Symmetric Channel I/O

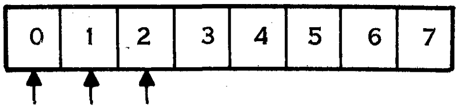
The basic supervisor call block shown in Section 5 applies to symmetric channel I/O, except as noted in the descriptions of the operations. The subset of sub-opcodes for symmetric channel I/O is as follows:

00	Open
01	Close
02	Close, Write EOF
05	Read Device Status
09	Symmetric Read
0B	Symmetric Write
0D	Write EOF

The following sub-opcodes perform operatons identical to those shown:

Sub-opcode	Operation	Identical to
0A	Read Direct	Read ASCII
0C	Write Direct	Write ASCII
03	Open and Rewind	Open
04	Close and Unload	Close
12	Open Extend	Open

The system flags (byte 4) in the supervisor call block apply to all symmetric channel I/O. These flags are:



2279685

Bit 0 — Busy flag. Set by system as follows:
1 — Busy.
0 — Operation completed.

Bit 1 — Error flag. Set by system as follows:
1 — Error.
0 — No error.

Bit 2 — End-of-file. Set by system as follows:
1 — An EOF mark was read.
0 — A read operation did not receive an EOF indication.

The user flags (byte 5) in the supervisor call block apply to all symmetric channel I/O. However, significance of these flags differs for various operations. The flags that apply to each operation are described in the detailed description of each operation.

The operations appropriate for symmetric IPC channels are described in subsequent paragraphs. The following sub-codes, which do not apply to symmetric channels, produce the indicated results:

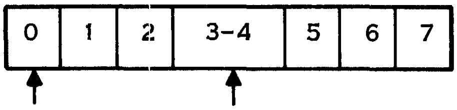
06	Ignored
07	Ignored
08	Ignored
0E	Ignored
0F	Ignored

8.3.2.1 Open. Sub-opcode >00 specifies an Open operation. Any task must perform an Open operation in order to access a channel. The owner task of a symmetric channel must perform an Open operation before any other Open operation is processed for a LUNO assigned to the channel. The access privileges of Open operations, subsequent to the first requestor open, are checked against the access privileges of the requestor opens to determine their validity. The access privileges of requestors are not checked against the access privileges of the channel owner. The access mode compatibility defined for device and file I/O applies also to IPC access modes.

The following fields of the basic supervisor call block apply to an Open operation:

- SVC opcode — >00
- Return code
- Sub-opcode — >00
- Logical unit number (LUNO)
- User flags
- <Data buffer address>
- <Read character count>

The following user flags apply to an Open operation:



2279686

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

Bits 3–4 — Access privilege flag. Set as follows:
00 — Exclusive write.
01 — Exclusive all.
10 — Shared.
11 — Read only.

The logical unit number (LUNO) field contains the LUNO assigned to the IPC channel to be opened.

Shared access is appropriate for most Open operations because it allows the calling task to both read and write, and it also allows other tasks to both read and write. Exclusive all access should not be used because it limits use of the channel to one task. The channel should be created as a nonshared channel when the channel is intended to serve only one requesting task.

The Open operation returns the device type code in the data buffer address field (bytes 6 and 7) of the supervisor call block. The device type code for a symmetric IPC channel is >8000.

When the calling task places zero in the read character count field (bytes 8 and 9) of the supervisor call block, the Open operation returns the maximum message length specified for the channel.

The following is an example of the source code for a supervisor call block to open a symmetric IPC channel:

```
OPENCH   DATA 0                OPEN LUNO >36 ASSIGNED TO A
         BYTE 0,>36            SYMMETRIC CHANNEL FOR SHARED
         BYTE 0                ACCESS
         BYTE >10
CHTYP    DATA0
MAXMSG   DATA 0
         DATA 0
```

8.3.2.2 Close. Sub-opcode >01 specifies a Close operation. The Close operation is required for a channel. When a requesting task closes the LUNO, the task has no further access to the channel.

If the owner task closes its LUNO assigned to the channel while LUNOs assigned by requesting tasks remain open, the channel becomes dormant. A dormant channel requires special processing before it is again available for normal use. To recover, each requesting task should close the LUNO assigned to the channel. When all requesting tasks have closed the channel, the owner task can open it, restoring the channel to normal use. A requesting task that issues a close request to the dormant channel can then open it. The Open operation remains queued and is actually performed after the owner task closes and then opens the channel.

This order of operations is unlikely; the capability for a dormant state is provided for cases when an owner task has unusual capabilities of error detection and recovery. Users who write application programs that interface with an owner task having these capabilities must be aware that the owner task uses these capabilities.

For IPC operations, the Close operation can return several error codes which are described in the *DNOS Messages and Codes Reference Manual.* The following hexadecimal error codes are returned most frequently.

>A6 Owner task has closed leaving the channel in the dormant state.

>A7 Owner task has aborted.

>E6 Requesting task on a nonshared symmetric channel has closed.

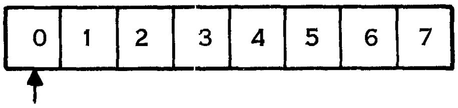
>E7 Requesting task on a nonshared symmetric channel as aborted.

For a nonshared channel, if the requesting task has closed its LUNO or has aborted, the owner task must issue a Close operation followed by an Open operation to be able to use the channel again.

The following fields of the basic supervisor call block apply to a Close operation:

- SVC code — 0
- Return code
- Sub-opcode — >01
- Logical unit number (LUNO)
- User flags

The following user flag applies to a Close operation:



2279687

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

The logical unit number (LUNO) field contains the LUNO to be closed.

The following is an example of the source code for a supervisor call block to close an IPC channel:

```
CLOCH    DATA 0              CLOSE LUNO >36 ASSIGNED TO
         BYTE 1,>36          SYMMETRIC CHANNEL
         DATA 0
         DATA 0
         DATA 0
         DATA 0
```

8.3.2.3 Close, Write EOF. The Close, Write EOF operation, sub-opcode >02, has the same effect as a Write EOF followed by a Close. The calling task can be suspended if there is no pending read operation to the channel.

8.3.2.4 Open and Rewind. The Open and Rewind operation, sub-opcode >03, is identical to the Open operation for an IPC channel.

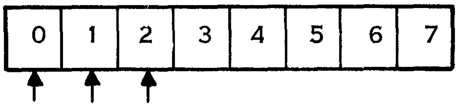
8.3.2.5 Close and Unload. The Close and Unload operation, sub-opcode >04, is identical to the Close operation for an IPC channel.

8.3.2.6 Symmetric Read. Sub-opcode >09 specifies a Symmetric Read operation. When there is a Symmetric Write operation pending for the channel, the Read operation transfers the message of the Write operation to the data buffer specified for the Read operation. When there is no pending Write operation, the calling task is suspended. The operation is completed and a message is stored in the data buffer when a Write operation is performed.

The following fields of the basic supervisor call block apply to a Symmetric Read operation:

- SVC opcode — >00
- Return code
- Sub-opcode — >09 (or >0A)
- Logical unit number (LUNO)
- <System flags>
- User flags
- Data buffer address
- Read character count
- <Actual read count>

The following system flags apply to a Symmetric Read operation:



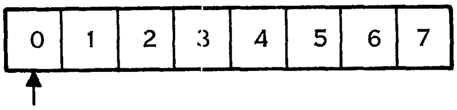
2279688

Bit 0 — Busy flag. Set by system as follows:
1 — Busy.
0 — Operation completed.

Bit 1 — Error flag. Set by system as follows:
1 — Error.
0 — No error.

Bit 2 — End-of-file. Set by system as follows:
1 — Write EOF operation performed by other task on the channel.
0 — Other task on the channel did not perform a Write EOF operation.

The following user flag applies to a Symmetric Read operation:



2279689

Bit 0 — Initiate flag. Set as follows:
 1 — System initiates the operation and returns control to the calling task.
 0 — System suspends the calling task until the operation has completed.

The logical unit number (LUNO) field contains the LUNO assigned to the IPC channel from which a message is to be read.

The data buffer address is the address of the buffer into which DNOS places the message.

The read character count is the length of the buffer.

DNOS returns the number of characters stored in the buffer in the actual read count field.

When the Write operation to which the Read operation is matched is a Write EOF operation, no data is transferred, the system sets the EOF flag in the system flags byte, and sets the actual read count to zero.

The following is an example of the source code for a supervisor call block for a Symmetric Read operation and code for the message buffer:

```
READCH   DATA 0                    READ MESSAGE FROM LUNO >36
         BYTE 9,>36                ASSIGNED TO SYMMETRIC CHANNEL
         BYTE 0,0
         DATA MBUFF
         DATA 80
         DATA 0
MBUFF    BSS 80                    MESSAGE BUFFER
```

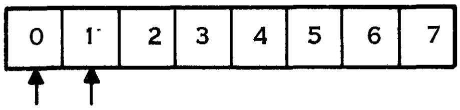
8.3.2.7 Symmetric Write. Sub-opcode >0B specifies a Symmetric Write operation. When there is a Symmetric Read operation pending on the channel, the operation transfers the message in the data buffer to the task that issued the Read operation. When there is no pending Read operation, the calling task is suspended. The operation completes and the message is transferred when a task issues a Read operation for the channel.

The following fields of the basic supervisor call block apply to a Symmetric Write operation:

- SVC code — 0
- Return code
- Sub-opcode — >0B (or >0C)
- Logical unit number (LUNO)

- <System flags>
- User flags
- Data buffer address
- Write character count

The following system flags apply to a Symmetric Write operation:

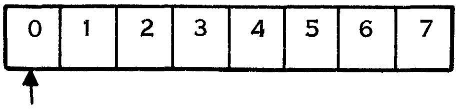


2279690

Bit 0 — Busy flag. Set by system as follows:
1 — Busy.
0 — Operation completed.

Bit 1 — Error flag. Set by system as follows:
1 — Error.
0 — No error.

The following user flag applies to a Symmetric Write operation:



2279691

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

The logical unit number (LUNO) field contains the LUNO assigned to the channel to which a message is to be written.

The data buffer address is the address of the buffer that contains the message to be written.

The write character count is the number of characters in the message.

The following is an example of the source code for a supervisor call block for a Symmetric Write operation:

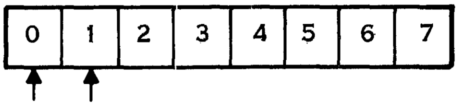
```
WRITCH    DATA 0                   WRITE MESSAGE TO LUNO >36
          BYTE >0B,>36             ASSIGNED TO CHANNEL
          BYTE 0,0
          DATA WRBUFF
          DATA 0
          DATA 8
WRBUFF    TEXT 'SEND TWO'
```

8.3.2.8 Write EOF. Sub-opcode >0D specifies a Write EOF operation. When there is a Symmetric Read operation pending on the channel, the operation sets the EOF flag and stores zero as the input record length for the Read operation. When there is no pending Read operation, the calling task is suspended. The operation completes when a task issues a Read operation for the channel.

The following fields of the basic supervisor call block apply to a Write EOF operation:

- SVC opcode — >00
- Return code
- Sub-opcode — >0D
- Logical unit number (LUNO)
- <System flags>
- User flags

The following system flags apply to a Write EOF operation:

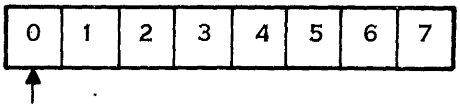


2279692

Bit 0 — Busy flag. Set by system as follows:
1 — Busy.
0 — Operation completed.

Bit 1 — Error flag. Set by system as follows:
1 — Error.
0 — No error.

The following user flag applies to a Write EOF operation:



2279693

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

The logical unit number (LUNO) field contains the LUNO assigned to the channel to which an EOF is to be written.

The following is an example of the source code for a supervisor call block for a Write EOF operation:

```
WESC     DATA 0                  WRITE EOF TO CHANNEL
         BYTE >D,>36             ASSIGNED TO LUNO >36
         BYTE 0,0
         DATA 0
         DATA 0
         DATA 0
```

When a Read operation matches a Write EOF, the end-of-file flag is set in the system flags of the Read.

8.3.3 Master/Slave Channel I/O

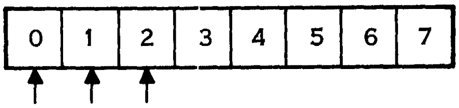
The basic supervisor call block shown in Section 5 applies to master/slave channel I/O, except as noted in the descriptions of the operations. Two sub-opcodes apply to both master and slave tasks, as follows:

00	Open
01	Close

Four additional operations are defined for master tasks:

05	Read Device Status
19	Master Read
1A	Read Call Block
1B	Master Write
1C	Redirect Assign LUNO

he system flags (byte 4) in the supervisor call block apply to all master/slave channel I/O. These lags are:



2279694

Bit 0 — Busy flag. Set by system as follows:
1 — Busy.
0 — Operation completed.

Bit 1 — Error flag. Set by system as follows:
1 — Error.
0 — No error.

Bit 2 — End-of-file. Set by system as follows:
1 — An EOF was read on the channel.
0 — A read operation did not read an EOF.

The user flags (byte 5) in the supervisor call block apply to all master/slave channel I/O. However, significance of these flags differs for various operations. The flags that apply to each operation are described in the detailed description of each operation.

The operations appropriate for owner tasks of master/slave channels are described in subsequent paragraphs. The following sub-opcodes, that do not apply to owner (master) tasks of master/slave channels, produce the indicated results:

02	Error
03	Error
04	Error
06	Error
07	Error
08	Error
09	Error
0A	Error
0B	Error
0C	Error
0D	Error
0E	Error
0F	Error

In general, the requesting (slave) tasks of a master/slave channel use the set of sub-opcodes defined for the default resource of the channel. The default resource type of channel expects that slaves will use only simple ASCII Read and ASCII Write requests. The master task determines which sub-opcodes apply and what occurs in response to each of them. The documentation of the master task must include this information and must be available to programmers of slave tasks.

An example of a typical sequence of operations performed by a master (owner) task to a master/slave channel is shown in Appendix C. In general, however, the sequence is as follows:

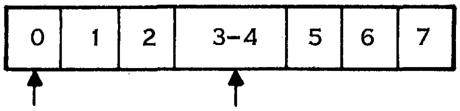
1. Assign LUNO to the channel. The LUNO should be a task-local LUNO.
2. Open the channel.
3. Issue a Master Read operation.
4. Process the requester operation returned in the Master Read data buffer.
5. Issue Master Write when processing of the requester operation is complete.
6. Repeat the Master Read/Master Write cycle as often as necessary.
7. Close the channel.
8. Release the LUNO to the channel.

8.3.3.1 Open. Sub-opcode >00 specifies an Open operation. The master task of a master/slave channel must perform an Open operation before any other Open operation may be processed for a LUNO assigned to the channel. The access privileges of Open operations, subsequent to the first requestor open, are checked against the access privileges of the requestor opens to determine their validity. The access privileges of the requestors are not checked against the privileges of the channel owner. The access mode compatibility defined for device and file I/O applies also to IPC access modes.

The following fields of the basic supervisor call block apply to an Open operation:

- SVC opcode — >00
- Return code
- Sub-opcode — >00
- Logical unit number (LUNO)
- User flags
- <Data buffer address>
- <Read character count>

The following user flags apply to an Open operation:



2279695

Bit 0 — Initiate flag. Set as follows:
- 1 — System initiates the operation and returns control to the calling task.
- 0 — System suspends the calling task until the operation has completed.

Bits 3-4 — Access privilege flag. Set as follows:
- 00 — Exclusive write.
- 01 — Exclusive all.
- 10 — Shared.
- 11 — Read only.

The logical unit number (LUNO) field contains the LUNO assigned to the IPC channel to be opened.

Shared access is appropriate for most Open operations because it allows the calling task to both read and write, and it also allows other tasks to both read and write. Exclusive all access should not be used because it limits use of the channel to one task. To restrict use of the channel to a single requesting (slave) task, the channel should be created as a nonshared channel.

The Open operation returns the device type code in the data buffer address field (bytes 6 and 7) of the supervisor call block. The device type code for a master/slave IPC channel is >8001 when the default resource type is IPC channel. Otherwise, the device type code returned when a calling task opens a LUNO assigned to a master/slave channel is the code for the default resource (file or device). The device type codes returned by an Open operation are listed in Table 8-2.

Table 8-2. Device/File Type Codes Returned by an Open Operation

Code	Device or File
0000	Dummy device
0001	ASR/KSR and teleprinter device (TPD)
0002	Printer
0003	Cassette unit
0004	Card reader
0005	911 VDT, 931 VDT, or 940 VDT
0006	Disk drive
0007	Communication devices
0008	Magnetic tape drive
003F	Special device
01FF	Sequential file
02FF	Relative record file
03FF	Key indexed file
04FF	Directory file
05FF	Program file
06FF	Image file

When the calling task places zero in the read character count field (bytes 8 and 9) of the supervisor call block, the Open operation returns the maximum message length specified for the channel.

Note that for a requester task, the maximum message length is the same as the maximum buffer length. For the master channel, the maximum message length is the maximum buffer length plus the length of the maximum Master Read Block.

The following source code example is a SVC call to open a master/slave IPC channel:

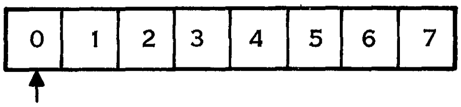
```
OPENCH   DATA 0                   OPEN LUNO >4C ASSIGNED TO MASTER/
         BYTE 0,>4C               SLAVE CHANNEL FOR SHARED ACCESS
         BYTE 0
         BYTE >10
MSCTYP   DATA 0
MSMML    DATA 0
         DATA 0
```

8.3.3.2 Close. Sub-opcode >01 specifies a Close operation. The Close operation is required for a channel. When a slave task closes the LUNO, the task has no further access to the channel.

The following fields of the basic supervisor call block apply to a Close operation:

- SVC code — 0
- Return code
- Sub-opcode — >01
- Logical unit number (LUNO)
- User flags

The following user flag applies to a Close operation:



2279696

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

The logical unit number (LUNO) field contains the LUNO to be closed.

The following is an example of the source code for the supervisor call block to close an IPC channel:

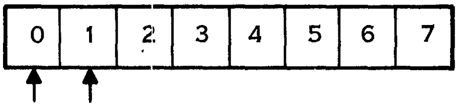
```
CLCHAN  DATA 0                        CLOSE LUNO >4C ASSIGNED TO
        BYTE 1,>4C                    MASTER/SLAVE CHANNEL
        DATA 0
        DATA 0
        DATA 0
        DATA 0
```

8.3.3.3 Read Device Status. Sub-opcode >05 specifies a Read Device Status operation. The Read Device Status operation returns the attributes of the channel to a specified buffer in the calling task.

The following fields of the basic supervisor call block apply to a Read Device Status operation:

- SVC code — 0
- Return code
- Sub-opcode — >05
- Logical unit number (LUNO)
- <System flags>
- User flags
- Data buffer address
- Read character count
- <Actual read count>

The following system flags apply to a Read Device Status operation:

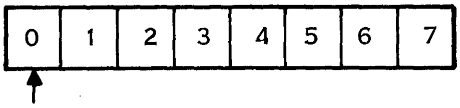


2279697

Bit 0 — Busy flag. Set by system as follows:
1 — Busy.
0 — Operation completed.

Bit 1 — Error flag. Set by system as follows:
1 — Error.
0 — No error.

The following user flag applies to a Read Device Status operation:



2279698

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

The logical unit number (LUNO) field contains the LUNO assigned to the IPC channel for which the attributes are required.

The data buffer address is the address of the buffer into which DNOS places the channel attributes.

The read character count is the length of the buffer. The count must be at least 10 to obtain all channel attributes. When a lesser value is entered, that number of bytes of information is returned. The higher numbered bytes are truncated.

DNOS returns the number of characters stored in the buffer in the actual read count field. This number is 10 or the read character count, whichever is less.

The attributes of an IPC channel consist of 10 bytes. The contents of the buffer following a Read Device Status operation are:

DEC	HEX		
0	0	CHANNEL FLAGS	
2	2	RESOURCE TYPE	RESOURCE TYPE FLAGS
4	4	MAXIMUM MESSAGE LENGTH	
6	6	ASSIGN COUNT	OPEN COUNT
8	8	RESERVED	

2279699

Byte	Contents
0–1	Channel flags, as follows: Bits 0–1 — Channel scope: 00 — Task-local. 01 — Job-local. 10 — Global.

Byte	Contents
	Bit 2 — Shared flag: 1 — Shared. 0 — Not shared.
	Bit 3 — Channel type flag: 1 — Not returned. 0 — Master/slave channel.
	Bit 4 — Master process assign flag: 1 — Master processes assigns. 0 — Master does not process assigns.
	Bit 5 — Master process abort flag: 1 — Master process aborts. 0 — Master does not process aborts.
	Bit 6 — Master process utility opcodes flag: 1 — Master processes utility opcodes. 0 — Master does not process utility opcodes.
	Bits 7-8 — Reserved.
	Bit 9 — Master open flag: 1 — Master task has issued open. 0 — Master task has not issued open.
	Bit 10 — Master close flag: 1 — Master task has issued close, or has aborted. 0 — Master task has not issued close.
	Bits 11-15 — Reserved.
2	Default resource type (see Create Channel operation).
3	Default resource type flags (see Create Channel operation).
4-5	Maximum message length. This is the maximum number of bytes of data transferred by a Master Read or Master Write operation. It includes space for the user's data buffer and all header and call block fields.
6	Assign count. Number of LUNOs currently assigned to the channel.
7	Open count. Number of slave tasks to which the channel is currently open.
8-9	Reserved.

The following is an example of the source code for a supervisor call block for a Read Device Status operation, and code for the buffer:

```
RDCHST   DATA 0           READ STATUS OF LUNO >4C ASSIGNED
         BYTE 5,>4C       TO MASTER/SLAVE CHANNEL
         BYTE 0,0
         DATA ATTR
         DATA 10
         DATA 0
ATTR     BSS 10           ATTRIBUTES BUFFER
```

8.3.3.4 Master Read. The Master Read operation is the means by which the master task receives the information required to process requests. IPC returns the requests issued by the slave tasks in response to Master Read operations.

After opening the channel, the master task issues a Master Read. If a request is pending, DNOS returns the information in the appropriate format as described in subsequent paragraphs. When the master task processes Assign LUNO operations, the first Master Read operation returns the Assign LUNO operation of the first slave task to use the channel.

When a request is not pending, a flag in the supervisor call block for the Master Read operation determines whether or not the master task is to be suspended. When the flag is set, the master task is not suspended but receives zero as the number of bytes read. Otherwise, the master task is suspended with the Master Read operation pending until a request is issued by a slave task. Then the master task is reactivated and receives the information about the requested operation in the specified buffer.

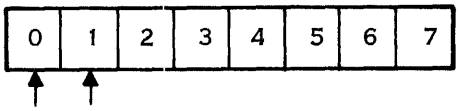
Sub-opcode >19 specifies a Master Read operation. The operation returns information from the supervisor call block, preceded by a header that contains information about the calling task. The master task may use the information in the header to identify the requesting task. The master task may alter the call block from the requesting task as appropriate for the requested processing but must not alter the contents of the header. This data should be used only for comparison with previous headers. It must not be used to access the requester's task space. The master task supplies data in a buffer when processing a read operation requested by a slave task except when zero characters are to be returned to the slave task. The data structure consisting of the header, the call block contents, and any required buffer is referred to as the master read buffer (MRB) in subsequent paragraphs. The master task must issue a master write request to return the MRB to the slave task.

The following fields of the basic supervisor call block apply to a Master Read operation:

- SVC code — 0
- Return code
- Sub-opcode — >19
- Logical unit number (LUNO)
- <System flags>

- User flags
- Data buffer address
- Read character count
- <Actual read count>

The following system flags apply to a Master Read operation:

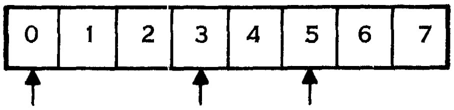


2279700

Bit 0 — Busy flag. Set by system as follows:
1 — Busy.
0 — Operation completed.

Bit 1 — Error flag. Set by system as follows:
1 — Error.
0 — No error.

The following user flags apply to a Master Read operation:



2279701

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

Bit 3 — Resolve logical names flag. Set as follows:
1 — The pathname in the MRB must have all logical names resolved.
0 — The original pathname specified by the slave task's call block is in the MRB.

Bit 5 — Do not suspend flag. Set as follows:
1 — Return control to the master task immediately, whether or not a request is pending.
0 — When no request is pending, suspend the master task until a request is issued.

Although it is valid to set both the initiate and the do not suspend flags for the same operation, it is not meaningful for both to be set; either flag alone returns control to the calling task.

The logical unit number (LUNO) field contains the LUNO assigned to the master/slave channel.

The data buffer address is the address of the buffer into which DNOS copies the MRB.

The read character count is the length of the buffer. The buffer should be as large as the maximum message length specified when the channel was created. When the buffer is not large enough to contain the MRB, as much of the data as the buffer can contain is returned, and the remainder is truncated.

DNOS returns the number of characters stored in the buffer in the actual read count field. This number is the length of the MRB and associated buffers or the read character count, whichever is less.

The MRB has several variations becâuse the supervisor call block within the MRB varies according to the requested operation and the default resource type. Three master/slave channel creation options affect the MRB the master task receives:

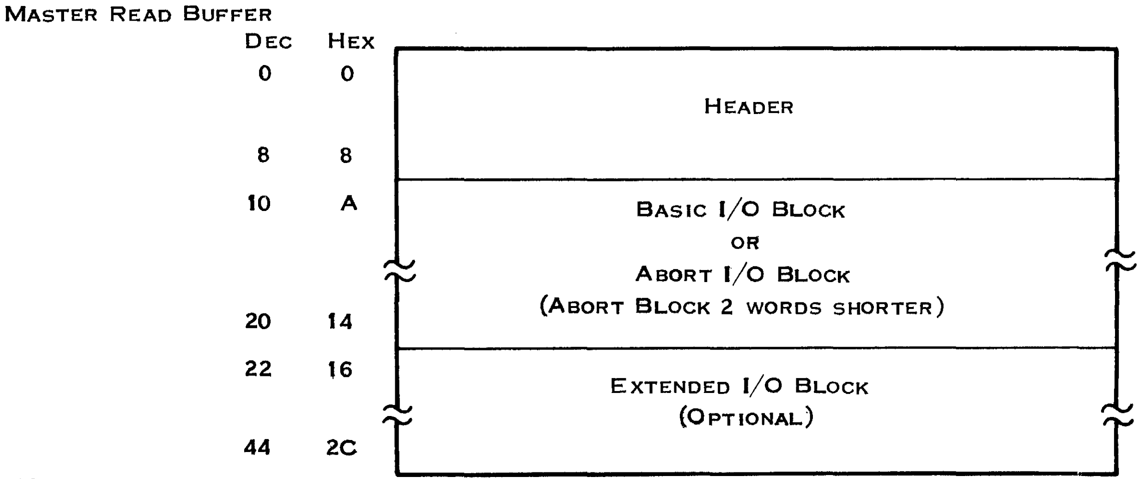
- Assign LUNO operations (bit 4 in channel flags)
- I/O Utility operations (bit 5 in channel flags)
- Abort I/O operations (bit 6 in channel flags)

When one of the above options is in effect, the master task receives an MRB that contains one of the following, dependent on option selected:

- Assign LUNO call block
- I/O Utility call block
- Abort I/O call block

The appropriate MRB is sent each time a slave task issues a request for Assign LUNO, I/O Utility operation, or Abort I/O operation.

The MRB consists of a header and either the basic I/O block (possibly with optional extended I/O block) or the abort I/O block. Either the basic I/O block or the extended I/O block must be specified. The components of the MRB are broken down as follows:



2283154

The contents of each component of the MRB are described in the following paragraphs.

CALLING TASK HEADER

DEC	HEX	
0	0	SYSTEM SECURITY DATA
2	2	CALL BLOCK ADDRESS
4	4	TASK STATUS BLOCK ADDRESS
6	6	JOB STATUS BLOCK ADDRESS
8	8	SYSTEM SECURITY INFORMATION

2283155

The calling task header of the MRB contains the following:

Byte	Contents
0–1	System security information.
2–3	Call block address. Address of the supervisor call block in the slave task.
4–5	Task status block address. Address of the task status block of the slave task.
6–7	Job status block address. Address of the job status block for the job to which the slave task belongs.
8–9	Security information.

Basic I/O Block

10	A	>00	<Return Code>
12	C	Sub–opcode	LUNO
14	E	<System Flags>	User Flags
16	10	Data Buffer Offset	
18	12	Read Character Count	
20	14	Write Character Count/<Actual Read Count>	

2283156

The basic I/O block of the MRB contains the following:

Byte	Contents
10	Opcode, >00.
11	Return code. The master task may return zero when the operation completes satisfactorily. When the operation completes in error, the task may return an error code.
12	Sub-opcode for desired operation. For Assign LUNO, >91.
13	Logical unit number (LUNO).
14	System flags. DNOS sets the busy flag (bit 0) before returning the call block to the master task.
15	User flags. User sets these flags for utility operations. The flags for those operations that use this field are specified and described in the paragraph on each operation.
16–17	Data buffer offset. Not used (reserved) for an Assign LUNO operation.
18–19	Read character count. Not used (reserved) for an Assign LUNO operation.
20–21	Write character count. Not used (reserved) for an Assign LUNO operation.

ABORT I/O BLOCK

10	A	>0F	<RETURN CODE>
12	C	USER FLAGS	LUNO
14	E	ABORTING TSB ADDRESS	
16	10	ABORTING JSB ADDRESS	

2283157

The abort I/O block of the MRB contains the following:

Byte	Contents
10	Opcode, >0F.
11	Return code. The master task may return zero when the operation completes satisfactorily. When the operation completes in error, the task may return an error code.
12	User flags. Set as follows: Bit 0 — Do not close flag. 1 — Do not close the LUNO if the task aborts. 0 — Close the LUNO if the task aborts. Bits 1-7 — Reserved.
13	Logical unit number (LUNO).
14–15	Aborting TSB address.
16–17	Aborting JSB address.

EXTENDED I/O BLOCK

22	16	KEY DEFINITION BLOCK OFFSET
24	18	RESERVED
26	1A	UTILITY FLAGS
28	1C	DEFINED LOGICAL RECORD LENGTH
30	1E	DEFINED PHYSICAL RECORD LENGTH
32	20	PATHNAME OFFSET
34	22	PARAMETER OFFSET
36	24	RESERVED
38	26	INITIAL FILE ALLOCATION
40	28	
42	2A	SECONDARY FILE ALLOCATION
44	2C	

2283158

The MRB contains the following:

Byte	**Contents**
22-23	Key definition block offset
24-25	Reserved.

Byte	Contents
26–27	Utility flags. User sets these flags for utility operations. The flags for each operation are specified and described in the paragraph on each operation.
28–29	Logical record length. Applies to Create operations.
30–31	Physical record length. Applies to Create operations.
32–33	Pathname offset. For all operations except Release LUNO, contains the relative address of a buffer that contains the pathname. The address is relative to byte 0 of the MRB. The format of the pathname buffer is: Byte 0 — Length n of pathname in bytes. Bytes1-n — Pathname.
34–35	Parameter offset. Address of the parameter buffer relative to byte 0 of the MRB. The parameters are logical name parameters, in the format described in the Name Management SVC description.
36–37	Reserved.
38–41	Initial file allocation. Applies to a Create operation.
42–45	Secondary file allocation. Applies to a Create operation for an expandable file.

The master task may return any currently defined DNOS error code for I/O SVCs listed in the *DNOS Messages and Codes Reference Manual.* A special code, >E5, has been reserved for special master task error reporting. This code indicates that the master task has detected an error. The master task should return additional information about the error in response to the next read request by the slave task. It is the responsibility of the master task to identify the slave task to which the error applies and to return appropriate information to that task.

Most requests include data buffers and pathname buffers; some include parameter buffers and other buffers. These buffers are placed in the MRB following the call block for the request. The offsets of these buffers in the MRB are addresses relative to byte 0 of the MRB. The actual number of buffers varies for each operation.

Other I/O utility operations use the same MRB. Notice that it is divided into three sections. Bytes 0 through 9 contain the header, and bytes 10 through 21 contain the basic I/O supervisor call block. These sections are common to all MRBs. MRBs for operations that use the basic I/O supervisor call block contain only these sections.

Relative File I/O Extension. Other operations add one or more additional words to the basic supervisor call block. For relative record file I/O, two words are added for the relative record number. The channel must have been created with a default resource type of relative record file. The additional words are as follows:

DEC	HEX	
22	16	RECORD NUMBER
24	18	

2279703

The extension contains the following:

Byte	Contents
22-25	Record number. The number of the record to be processed in a relative record file.

Key Indexed File I/O Extension. For key indexed file I/O, the following words are used in the extended I/O block. The channel must be created with a default resource type of key indexed file. The additional words are as follows:

22	16	CURRENCY BLOCK ADDRESS	
24	18	RESERVED	
26	1A	RESERVED	
28	1C	CURRENCY BLOCK	
46	2E		

2283211

The extension contains the following:

Byte	**Contents**
22-23	Currency block address.
24-25	Reserved.
26-27	Reserved.
28-46	Currency block (contains 20 bytes).

Write with Reply Extension. A Write with Reply operation requires two additional words and the reply block. The channel must have been created with a default resource type of a device for which a Write with Reply operation is valid. The additional words are as follows:

DEC	HEX	
22	16	REPLY BLOCK OFFSET
24	18	[RESERVED]
26	1A	REPLY BUFFER OFFSET
28	1C	REPLY CHARACTER COUNT
30	1E	<ACTUAL REPLY COUNT>

2279704

The extension contains the following:

Byte	**Contents**
22-23	Reply block offset, relative to byte 0 of the MRB. Specifically, >1A.
24-25	[Reserved].
26-27	Reply buffer offset, relative to byte 0 of the MRB. The reply is placed in the buffer by the master task.
28-29	Reply character count. The length of the reply buffer, set by the requester task.
30-31	Actual reply count. The number of characters in the reply placed in the reply buffer by the master task.

Read Extension. A Read operation requested by a slave task may request validation by setting the character validation flag in the extended user flags field. When validation is required, a two-word extension is provided, as follows:

DEC	HEX	
22	16	VALIDATION BUFFER OFFSET
24	18	[RESERVED]

2279705

Byte	**Contents**
22-23	Validation buffer offset, relative to byte 0 of the MRB. The buffer contains data that defines the validation.
24-25	[Reserved].

Extension for Resource-Specific I/O to a VDT. Another extension is required for resource-specific I/O to a video display terminal (VDT). The channel must have been created with a default resource type of 911 VDT. The required extension consists of the following five words:

DEC	HEX		
22	16	ZERO OR REPLY BLOCK OR VALIDATION OFFSET	
24	18	EXTENDED USER FLAGS	
26	1A	FILL CHARACTER	<EVENT BYTE>
28	1C	CURSOR POSITION ROW	COLUMN
30	1E	FIELD BEGINNING ROW	COLUMN

2279706

The extension contains the following:

Byte	Contents
22–23	Zero, for operations that require neither reply nor validation. Reply block offset or validation offset, as appropriate. Each offset is relative to byte 0 of the MRB.
24–25	Extended user flags. These flags define resource-specific VDT I/O.
26	Fill character.
27	Event byte. Byte in which master task returns event character.
28	Cursor position row.
29	Cursor position column.
30	Field beginning definition row.
31	Field beginning definition column.

When character validation is required with resource-specific I/O to the VDT, the word in bytes 22 and 23 contains the address of the validation buffer relative to byte 0 of the MRB, and a word is added to the MRB at the end of the VDT extension as follows:

DEC	HEX	
32	20	[RESERVED]

2279707

For a Write with Reply operation to the VDT in the resource-specific I/O mode, the word in bytes 22 and 23 contains the address of the reply block relative to byte 0 of the MRB. The address is >22. An additional word and the reply block are added to the MRB at the end of the VDT extension as follows:

DEC	HEX	
32	20	[RESERVED]
34	22	REPLY BUFFER OFFSET
36	24	REPLY CHARACTER COUNT
38	26	<ACTUAL REPLY COUNT>

2279708

The extension contains the following:

Byte	Contents
32-33	[Reserved].
34-35	Reply buffer offset, relative to byte 0 of the BRB. The reply is placed in the buffer.
36-37	Reply character count. The length of the reply buffer, supplied by the slave task.
38-39	Actual reply count. The number of characters in the reply placed in the reply buffer, supplied by the slave task.

Direct Disk I/O Extension. For direct disk I/O two methods are used to access the desired file. To access the file by track address and sector number, the following words are required in the extended I/O block.

22	16	TRACK ADDRESS	
24	18	SECTORS/RECORD	SECTOR NUMBER

2283212

This extension contains the following:

Byte	Contents
22-23	Track address. The number of the track to access in the disk.
24	Number of sectors per record.
25	Sector number.

Alternatively, to access the file by ADU number and sector offset use the following words in the extended I/O block.

22	16	ADU NUMBER
24	18	SECTOR OFFSET INTO ADU

2283213

This extension contains the following:

Byte	Contents
22–23	Number of the ADU to access.
24–25	Sector offset into the specified ADU.

The following is an example of the source code for a supervisor call block for a Master Read operation, and code for the buffer:

```
MREAD     DATA 0                   MASTER READ OF LUNO >4C ASSIGNED
          BYTE >19,>4C             TO MASTER/SLAVE CHANNEL,
          BYTE 0,0                 SUSPEND WHEN NO REQUEST IS
          DATA MRB                 PENDING
          DATA 100
          DATA 0
MRB       BSS 100                  REQUEST BUFFER
```

8.3.3.5 Read Call Block. The Read Call Block operation is similar to the Master Read operation, but has two important differences. Every Master Read operation returns the entire MRB to the master task; it must be followed by a Master Write operation. The Read Call Block operation returns the first 24 bytes of the MRB (no buffers) and is not followed by a Master Write operation. The first 24 bytes of the MRB consist of the 10 bytes of the header, the 12 bytes of the basic I/O supervisor call block, and two bytes of the extension, if any. A Read Call Block operation may be followed by any required processing.

The master task may issue a Read Call Block request to obtain information about a request from a slave task without being committed to process the request and return results to the task.

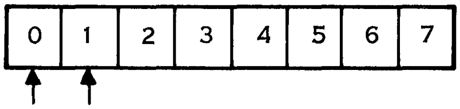
Sub-opcode >1A specifies a Read Call Block operation. When there is no request from a slave task pending and the do not suspend flag is set, the operation returns zero as the number of characters read and the master task resumes execution. When there is no request from a slave task pending and the do not suspend flag is not set, the master task is suspended until there is a request from a slave task.

The following fields of the basic supervisor call block appy to a Read Call Block operation:

- SVC code — 0
- Return code
- Sub-opcode — >1A
- Logical unit number (LUNO)
- <System flags>
- User flags
- Data buffer address

- Read character count
- <Actual read count>

The following system flags apply to a Read Call Block operation:

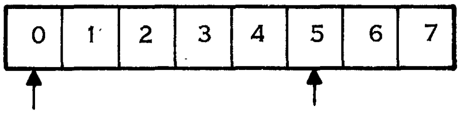


2279709

Bit 0 — Busy flag. Set by system as follows:
1 — Busy.
0 — Operation completed.

Bit 1 — Error flag. Set by system as follows:
1 — Error.
0 — No error.

The following user flags apply to a Read Call Block operation:



2279710

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

Bit 5 — Do not suspend flag. Set as follows:
1 — Return control to the master task immediately, whether or not a request is pending.
0 — When no request is pending, suspend the master task until a request is issued.

Although it is valid to set both the initiate and the do not suspend flags for the same operation, it is not meaningful for both to be set; either flag alone returns control to the calling task.

The logical unit number (LUNO) field contains the LUNO assigned to the master/slave channel.

The data buffer address is the address of the buffer into which DNOS copies the partial MRB.

The read character count is the length of the buffer. The buffer should consist of at least 24 bytes. Otherwise, only the specified number of bytes of the MRB is returned.

DNOS returns the number of characters stored in the buffer in the actual read count field. This number is 24 or the read character count, whichever is less.

The following is an example of the source code for a supervisor call block for a Read Call Block operation and code for the buffer:

```
RCB     DATA 0              READ CALL BLOCK FOR LUNO >4C
        BYTE >1A,>4C        ASSIGNED TO MASTER/SLAVE
        BYTE 0,0            CHANNEL, SUSPEND WHEN NO
        DATA PMRB           REQUEST IS PENDING
        DATA 24
        DATA 0
PMRB    BSS 24              REQUEST BUFFER
```

8.3.3.6 Master Write. The Master Write operation matches the Master Read operation; the master task obtains a request with a Master Read, performs appropriate processing, and returns the request (updated to show the results of the operation) with a Master Write operation.

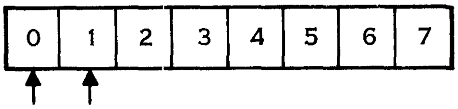
Depending upon the type of operation requested, the entire message returned by the Master Read operation may not have to be returned by the Master Write. When the slave task requests a Read, the entire MRB with the data buffer must be returned. For many operations, the header and the first two bytes of the call block (including the return code) are adequate.

Sub-opcode >1B specifies a Master Write operation. The MRB being written must match an MRB that was sent to the master task. Otherwise, DNOS returns an error code. The same error code (>A2) is returned when the requesting task has issued an Abort I/O SVC to the channel while the master task is processing the requesting task's MRB.

The following fields of the basic supervisor call block apply to a Master Write operation:

- SVC opcode — >00
- Return code
- Sub-opcode — >1B
- Logical unit number (LUNO)
- <System flags>
- User flags
- Data buffer address
- Write character count

The following system flags apply to a Master Write operation:

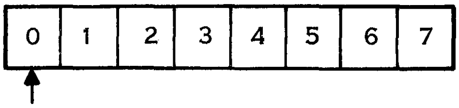


2279711

Bit 0 — Busy flag. Set by system as follows:
1 — Busy.
0 — Operation completed.

Bit 1 — Error flag. Set by system as follows:
1 — Error.
0 — No error.

The following user flag applies to a Master Write operation:



2279712

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

The logical unit number (LUNO) field contains the LUNO assigned to the master/slave channel.

The data buffer address is the address of the buffer that contains the MRB. The buffer must contain the entire header and at least two bytes of the call block from the requesting task.

The write character count is the number of characters of the MRB to be returned.

A master task receives a special MRB when a requesting task terminates abnormally. The task termination process sends a special block that has a Close operation sub-opcode (>01) and a call block address of >FFFF. The master task must be able to recognize this as a termination message, perform any processing required for correct operation of the master task, and return the MRB using a Master Write operation.

The following is an example of the source code for a supervisor call block for a Master Write operation:

```
MW     DATA 0              MASTER WRITE FOR CHANNEL ASSIGNED
       BYTE >1B,>4C        TO LUNO >4C
       BYTE 0,0
       DATA MRB
       DATA 0
       DATA 24
```

8.3.3.7 Redirect Assign LUNO. A channel master task that processes Assign LUNO operations can perform a Redirect Assign LUNO operation as an alternative to a Master Write. The operation is only valid when the requester's operation causes the IPC subsystem to redirect the requester's Assign LUNO operation to another I/O resource.

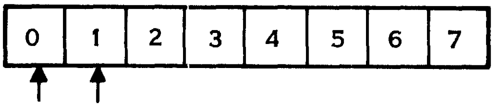
The master task specifies the target I/O resource by placing the resource name in a pathname buffer in the MRB. The pathname buffer should be of the same format as a pathname buffer for a utility operation (the first byte in the buffer contains the number of characters in the pathname). The pathname pointer in the Assign LUNO call block should be changed to point to the target I/O resource pathname buffer.

Sub-opcode >1C specifies a Redirect Assign LUNO operation. The MRB being written must match an MRB that was sent to the master task. Otherwise, DNOS returns an error code. The same error code (>A2) is returned when the requesting task has issued an Abort I/O SVC to the channel while the master task is processing the requesting task's MRB.

The following fields of the basic supervisor call block apply to a Redirect Assign LUNO operation:

- SVC opcode — >00
- Return code
- Sub-opcode — >1C
- Logical unit number (LUNO)
- <System flags>
- User flags
- Data buffer address
- Write character count

The following system flags apply to a Redirect Assign LUNO operation:

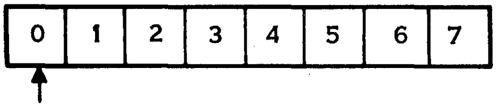


2283196

Bit 0 — Busy flag. Set by system as follows:
1 — Busy.
0 — Operation completed.

Bit 1 — Error flag. Set by system as follows:
1 — Error.
0 — No error.

The following user flag applies to a Redirect Assign LUNO operation:



2283190

Bit 0 — Initiate flag. Set as follows:
1 — System initiates the operation and returns control to the calling task.
0 — System suspends the calling task until the operation has completed.

The logical unit number (LUNO) field contains the LUNO assigned to the master/slave channel.

The data buffer address is the address of the buffer that contains the MRB. The buffer must contain the entire header and the entire call block from the requesting task.

The write character count is the number of characters of the MRB to be returned.

The following is an example of the source code for a supervisor call block for a Redirect Assign LUNO operation:

```
MW      DATA 0                  REDIRECT ASSIGN LUNO FOR CHANNEL
        BYTE >1C,>4C            ASSIGNED TO LUNO >4C
        BYTE 0,0
        DATA MRB
        DATA 0
        DATA 100
```

8.3.4 Master/Slave Channel Example

An example illustrating the typical sequence of master/slave channel operations performed by the master task to process a request from a slave task is shown in Appendix C.

9

Volume Management

9.1 DISK VOLUMES

Any disk medium used on a disk drive is called a disk volume. A disk volume must be initialized before being used in the system. A disk volume that contains obsolete information may be reinitialized and used in the system as if it were a new volume. A previously initialized volume must be installed in the system each time it is physically mounted in the disk drive, and it must be unloaded from the system before another volume can be installed in the drive.

9.2 INITIALIZING A NEW VOLUME

Initializing a new volume consists of the following operations:

- Optionally formatting all tracks on the disk.
- Writing volume overhead data on sector 0, track 0.
- Writing the list of bad allocatable disk units (ADUs) on sector 1, track 0.
- Writing partial bit maps (showing ADU availability) on the remaining sectors of track 0.
- Writing the volume directory on the disk.
- Building an .S$DIAG file
- Optionally writing a loader on track 1.
- Issuing an Install Volume SVC to install the new volume.

The Initialize New Disk Volume SVC (opcode >38) may be issued by software privileged tasks to perform these functions. The disk must not be hardware write-protected.

Formatting the tracks on the disk destroys any data previously stored on the disk. Therefore, do not issue this SVC for a volume that contains any data that is currently required and that does not exist on another volume.

The supervisor call block for the SVC is as follows:

SVC >38 -- Initialize New Disk Volume

Align on word boundary
Privileged task only

Dex	Hex		
0	0	>38	<Return Code>
2	2	Disk Drive Name	
4	4		
6	6	Volume Name	
12	C		
14	E	Number of Directory Entries	
16	10	Bad Track LUNO	Flags
18	12	Default Physical Record Size	
20	14	Hardware Interleaving Factor	
22	16	Loader LUNO	Reserved

2279713

The call block contains the following:

Byte **Contents**

0 Opcode, >38.

1 Return code. DNOS returns zero when the operation completes satisfactorily. When the operation completes in error, DNOS returns an error code.

2-5 Device name of disk drive (ASCII characters).

6-13 Volume name. One through eight ASCII characters (valid pathname characters)

14-15 Number of entries in volume directory. The actual number of entries in the volume directory is the lowest prime number equal to or greater than the number specified.

16 LUNO assigned to bad track file, or zero. Enter zero when there is no bad track file. The bad track list on track 0 sector 1 will be combined with the bad tracks in the bad track file.

17 Flags:
Bit 0 — Set to 1.
Bit 1 — Set to 1 when loader is to be written on track 1. Set to 0 when no loader is to be written.
Bit 2 — Set to 1 when the disk is not to be formatted. Set to 0 when the disk is to be formatted.
Bits 3-7 — Reserved.

18-19 Default physical record size for files on this volume, or zero. Enter a zero to specify default value supplied for the disk drive during system generation.

20-21 Hardware interleaving factor used in formatting the disk. Range is 0 through n, where n is the number of sectors per track minus 2. Enter zero for factor of 1 used for double-sided, double-density diskettes.

22 LUNO assigned to file that contains a loader in image format, or zero. Enter zero to specify system default loader. This byte is ignored unless flag byte contains >C0.

23 Reserved.

Disk packs are shipped with a list of bad tracks. This list must be written on a file, and the file assigned to a LUNO. The LUNO is entered in byte 16 of the supervisor call block. The format of the bad track file is:

```
Head, Cylinder;
Head, Cylinder;
        .
        .
        .

        or

Head, Cylinder; . . .   Head, Cylinder;
```

Follow the last entry with a blank line.

The loader options allow the system default loader or a task-supplied loader to be written on track 1. A disk that is not a system disk and does not require a loader may be initialized with no loader on track 1. When the task supplies a loader, it must be an image file of the loader, and the file must be assigned to the LUNO specified in byte 22.

The following is an example of coding for a supervisor call block for an Initialize New Disk Volume SVC:

```
INIT    EVEN                    INITIALIZE DISK VOLUME IN DRIVE
        BYTE >38                DS04 WITH 20 ENTRIES IN VCATALOG.
        BYTE 0                  BAD TRACK FILE LUNO IS >3D AND
        TEXT 'DS04'             LOADER IMAGE FILE LUNO IS >2F.
        TEXT 'NEWVOL  '         VOLUME NAME IS NEWVOL AND DEFAULTS
        DATA 20                 FOR PHYSICAL RECORD SIZE AND
        BYTE >3D                INTERLEAVING APPLY.
        BYTE >C0
        DATA 0
        DATA 0
        BYTE >2F
        BYTE 0
```

9.3 INSTALLING A VOLUME

Installing a disk volume associates the volume name with the disk drive device name. When a disk volume is physically installed, enter an Install Disk Volume SVC (opcode >20) to provide the volume name to the system. Specifically, the SVC:

- Verifies that the specified drive is available.
- Verifies that the specified volume is mounted on the drive.
- Initializes memory required for the disk manager.
- Deletes any temporary files on VCATALOG on the disk.

The supervisor call block for the SVC is as follows:

SVC > 20 -- INSTALL DISK VOLUME

ALIGN ON WORD BOUNDARY
PRIVILEGED TASK ONLY

DEC	HEX		
0	0	>20	< RETURN CODE>
2	2	DISK DRIVE NAME	
4	4		
6	6	VOLUME NAME	
12	C		

2279714

The call block contains the following:

Byte	Contents
0	Opcode, >20.
1	Return code. DNOS returns zero when the operation completes satisfactorily. When the operation completes in error, DNOS returns an error code.
2-5	Device name of disk drive (ASCII characters).
6-13	Volume name. One through eight ASCII characters. When another volume is installed on the specified disk drive, DNOS returns an error code in byte 1 and the volume name of the installed volume in this field.

The following is an example of coding for a supervisor call block for an Install Disk Volume SVC:

```
         EVEN                        INSTALL VOLUME NEWVOL IN DISK DRIVE
INSVOL   BYTE >20                    DS02.
         BYTE 0
         TEXT 'DS02'
         TEXT 'NEWVOL  '
```

9.4 UNLOADING A VOLUME

Unloading a volume consists of advising the system that the volume is no longer available. The Unload Disk Volume SVC (opcode>34) should be issued before the disk is physically removed from the drive. Specifically, the unload SVC:

- Determines the device on which the specified volume is installed.
- Verifies that no LUNOs are currently assigned to any files in the volume.
- Releases all memory allocated to support this volume.
- Updates the physical device table (PDT) for the disk drive to show that the volume is unloaded.
- Writes device statistics to the system log.
- Returns the disk drive device name.

The Unload Disk Volume SVC cannot unload the system disk.

The supervisor call block for the SVC is as follows:

SVC >34 -- UNLOAD DISK VOLUME

ALIGN ON WORD BOUNDARY
PRIVILEGED TASK ONLY

DEC	HEX		
0	0	>34	<RETURN CODE>
2	2	<DISK DRIVE NAME>	
4	4		
6	6	VOLUME NAME	
12	C		

2279715

The call block contains the following:

Byte	Contents
0	Opcode, >34.
1	Return code. DNOS returns zero when the operation completes satisfactorily. When the operation completes in error, DNOS returns an error code.
2–5	DNOS returns the device name of the disk drive on which the specified volume was installed.
6–13	Volume name. One through eight ASCII characters.

The following is an example of coding for a supervisor call block for an Unload Disk Volume SVC:

```
        EVEN                        UNLOAD VOLUME OLDVOL.
UNLVOL  BYTE >34
        BYTE 0
DISKID  BSS 4
        TEXT 'OLDVOL  '
```

10

Task Support

10.1 TASK SUPPORT FUNCTIONS

The group of supervisor calls described in this section provides the following support services to all tasks:

- Data conversion
- Encryption and decryption of data
- Job accounting
- Memory control
- Task synchronization
- Supplying status and system information

10.2 DATA CONVERSION

When numbers are entered, the ASCII characters must be converted to their binary equivalent for processing. Similarly, binary values must be converted to ASCII characters for output. DNOS provides supervisor calls to perform conversions to and from decimal ASCII numbers and hexadecimal ASCII numbers.

10.2.1 Converting Binary Data to Decimal ASCII

When a binary value is to be printed, it must be converted to ASCII characters. The Convert Binary to Decimal ASCII SVC (opcode >0A) converts the binary contents of a word to decimal digits and returns the ASCII characters corresponding to the digits to the calling task. The SVC converts the contents of workspace register 0, placing the ASCII codes for the sign and for the digits in the supervisor call block.

The supervisor call block for the SVC is as follows:

SVC > 0A -- CONVERT BINARY TO DECIMAL ASCII

DEC	HEX		
0	0	>0A	<RETURN CODE>
2	2	<ASCII MINUS/BLANK>	<ASCII DIGIT>
4	4	<ASCII DIGIT>	<ASCII DIGIT>
6	6	<ASCII DIGIT>	<ASCII DIGIT>

2279716

The call block contains the following:

Byte	Contents
0	Opcode, >0A.
1	Return code.
2-7	ASCII field. DNOS returns six characters. When the result is negative, byte 2 contains a minus sign. Otherwise byte 2 contains a blank. The digits are placed in bytes 3 through 7, right justified, zero suppressed with leading blanks.

The following table shows the results of converting several binary values:

Register	Supervisor Call Block Bytes					
0	2	3	4	5	6	7
>0001	>20	>20	>20	>20	>20	>31
>7FFF	>20	>33	>32	>37	>36	>37
>FFFF	>2D	>20	>20	>20	>20	>31

The following is an example of coding for a supervisor call block for a Convert Binary to Decimal ASCII SVC:

```
CBNDEC  BYTE >0A          CONVERT VALUE IN R0 TO DECIMAL
        BYTE 0
VALUE   BYTE 0,0
        BYTE 0,0
        BYTE 0,0
```

10.2.2 Converting Decimal ASCII to Binary Data

A number entered at a terminal or supplied to the task in some other way as a group of ASCII characters must be converted to its binary equivalent for processing. When the number is a decimal number, the Convert Decimal ASCII to Binary SVC (opcode >0B) may be called to perform the conversion. The ASCII characters to be converted are placed in the supervisor call block, and the binary result is returned in workspace register 0. The range of values that may be converted is minus 32,768 through 32,767.

The supervisor call block for the SVC is as follows:

SVC > 0B -- CONVERT DECIMAL ASCII TO BINARY

DEC	HEX		
0	0	>0B	<RETURN CODE>
2	2	ASCII SIGN	ASCII DIGIT
4	4	ASCII DIGIT	ASCII DIGIT
6	6	ASCII DIGIT	ASCII DIGIT

2279717

The call block contains the following:

Byte	Contents
0	Opcode, >0B.
1	Return code. DNOS returns zero when the operation completes satisfactorily. When the operation completes in error, DNOS returns an error code.
2	ASCII sign. Allowable entries are: >2B — ASCII plus sign. >2D — ASCII minus sign. >20 — ASCII blank. Appropriate when the number is positive or zero. >30 — ASCII zero. Appropriate when the number is positive or zero.
3-7	ASCII characters that represent the number. The number must be right justified, with leading zeros or leading blanks. Embedded blanks are not allowed.

The following table shows several values, how they are represented in the supervisor call block, and the results of conversion in workspace register 0 and in byte 1 of the block:

Value	**Supervisor Call Block Bytes**						**Register**	**Byte**
0	**2**	**3**	**4**	**5**	**6**	**7**	**0**	**1**
+00001	>2B	>30	>30	>30	>30	>31	>0001	0
1	>20	>20	>20	>20	>20	>31	>0001	0
1A	>20	>20	>20	>20	>31	>41	UNDEF.	>FF
032767	>30	>33	>32	>37	>36	>37	>7FFF	0
-00001	>2D	>30	>30	>30	>30	>31	>FFFF	0

The following is an example of coding for a supervisor call block for a Convert Decimal ASCII to Binary SVC:

```
CDECBN   BYTE >0B                 CONVERT VALUE IN BYTES 2 THROUGH 7
         BYTE 0                   TO BINARY AND PLACE RESULT IN R0.
VAL      BYTE 0,0
         BYTE 0,0
         BYTE 0,0
```

The code in the example initializes bytes 2 through 7 to zero. Other code would move ASCII characters into this field before executing the call.

10.2.3 Converting Binary Data to Hexadecimal ASCII

When a binary value is to be printed as hexadecimal, it must be converted to ASCII characters representing the hexadecimal value. The Convert Binary to Hexadecimal ASCII SVC (opcode >0C) performs this conversion. The value in workspace register 0 is converted and the results are placed in the supervisor call block.

The supervisor call block for the SVC is as follows:

SVC >0C -- CONVERT BINARY TO HEXADECIMAL ASCII

DEC	HEX		
0	0	>0C	<RETURN CODE>
2	2	<ASCII DIGIT>	<ASCII DIGIT>
4	4	<ASCII DIGIT>	<ASCII DIGIT>

2279718

The call block contains the following:

Byte	Contents
0	Opcode, >0C.
1	Return code.
2-5	ASCII field. DNOS returns four characters.

The following table shows the results of converting several binary values:

Register	Supervisor Call Block Bytes			
0	2	3	4	5
>0001	>30	>30	>30	>31
>7FFF	>37	>46	>46	>46
>FFFF	>46	>46	>46	>46

The following is an example of coding for a supervisor call block for a Convert Binary to Hexadecimal ASCII SVC:

```
CBNHEX   BYTE >0C                   CONVERT VALUE IN R0 TO HEXADECIMAL
         BYTE 0
HVAL     BYTE 0,0
         BYTE 0,0
```

10.2.4 Converting Hexadecimal ASCII to Binary Data

Hexadecimal digits entered at a terminal or supplied to the task in some other manner as ASCII characters must be converted to an equivalent binary word for further processing. The Convert Hexadecimal ASCII to Binary SVC (opcode >0D) performs that function. The characters to be converted are placed in the supervisor call block, and the result is returned in workspace register 0.

The supervisor call block for the SVC is as follows:

SVC >0D -- CONVERT HEXADECIMAL TO BINARY

DEC	HEX		
0	0	>0D	<RETURN CODE>
2	2	ASCII DIGIT	ASCII DIGIT
4	4	ASCII DIGIT	ASCII DIGIT

2279719

The call block contains the following:

Byte	Contents
0	Opcode, >0D.
1	Return code. DNOS returns a zero in this byte when the operation completes satisfactorily. When the operation completes in error, DNOS returns an error code.
2-5	The ASCII characters that represent the number.

The following table shows several values, how they are represented in the supervisor call block, and the results of conversion in workspace register 0 and in byte 1 of the block:

	Supervisor Call Block Bytes				Register	Byte
1	2	3	4	5	0	1
>0001	>30	>30	>30	>31	>0001	0
>003K	>30	>30	>33	>4B	UNDEF.	>FF
>7FFF	>37	>46	>46	>46	>7FFF	0
>FFFF	>46	>46	>46	>46	>FFFF	0

The following is an example of coding for a supervisor call block for a Convert Hexadecimal to Binary SVC:

```
SCBK     BYTE >0D                  CONVERT VALUE IN BYTES 2 THROUGH 5
         BYTE 0                    TO BINARY AND PLACE RESULT IN R0.
VAL1     BYTE 0,0
         BYTE 0,0
```

The code in the example initializes bytes 2 through 5 to zero. Other code would move ASCII characters into this field before executing the call.

10.3 ENCRYPTING AND DECRYPTING OF DATA

A user task may call upon DNOS to encrypt and decrypt data for any purpose. Records of files, IPC messages, or data of any type can be encrypted. The Get Encrypted Value SVC is available to encode the records of files, and the Get Decrypted Value SVC may be used to decode an encrypted record. The encryption algorithm does not provide a high level of security, but does provide a degree of privacy for the data. When more sophisticated encryption is required to provide a greater degree of security or to communicate with other systems, these SVCs should not be used. The user must either write programs to encrypt and decrypt the data, or supply SVCs for that purpose.

System generation must include the optional encryption SVC group for these operations to function. The encryption SVCs are included when generating file security and when requesting the optional SVC group named ENCRYPTION.

10.3.1 Encrypting Data

The Get Encrypted Value SVC (opcode >45) encrypts the contents of a specified buffer using an encryption key supplied in the supervisor call block. The SVC replaces the contents of the buffer with the encrypted version. The encrypted version is different when a different encryption key is used. For accurate decrypting of the data, the Get Decrypted Value SVC must use the same encryption key.

The supervisor call block for the SVC is as follows:

SVC > 45 -- GET ENCRYPTED VALUE ALIGN ON WORD BOUNDARY

DEC	HEX		
0	0	>45	<RETURN CODE>
2	2	ENCRYPTION KEY	
8	8		
10	A	NUMBER OF BYTES OF DATA	
12	C	BUFFER ADDRESS	
14	E	RESERVED	

2279720

The call block contains the following:

Byte	Contents
0	Opcode, >45.
1	Return code. DNOS returns a zero in this byte when the operation completes satisfactorily, or an error code when the operation completes in error.
2-9	Encryption key. May contain any eight bytes of data which will be used by the system in the algorithm for encrypting the data. This key must be used to decrypt the data.
10-11	Number of bytes of data. The length of the data to be encrypted.
12-13	Buffer address. Address of buffer that contains the data to be encrypted.
14-15	Reserved.

The following is an example of coding for a supervisor call block for a Get Encrypted Value SVC:

```
        EVEN                       ENCRYPT DATA IN BUFFER MSG2
GEV     BYTE >45
        BYTE 0
        TEXT '34544658'
        DATA 42
        DATA MSG2
        DATA 0
```

10.3.2 Decrypting Data

The Get Decrypted Value SVC (opcode >46) decrypts the contents of a specified buffer using an encryption key supplied in the supervisor call block. The SVC replaces the contents of the buffer with the decrypted version. The encryption key must be the key with which the data was encrypted using the Get Encrypted Value SVC.

The supervisor call block for the SVC is as follows:

SVC > 46 -- GET DECRYPTED VALUE   ALIGN ON WORD BOUNDARY

DEC	HEX		
0	0	>46	<RETURN CODE>
2	2	ENCRYPTION KEY	
8	8		
10	A	NUMBER OF BYTES OF DATA	
12	C	BUFFER ADDRESS	
14	E	RESERVED	

2279721

The call block contains the following:

Byte **Contents**

0 Opcode, >46.

1 Return code. DNOS returns a zero in this byte when the operation completes satisfactorily, or an error code when the operation completes in error.

2-9 Encryption key. Must contain the eight bytes of data that was used to encrypt the data.

10-11 Number of bytes of data. The length of the data to be decrypted.

Byte	Contents
12–13	Buffer address. Address of buffer that contains the data to be decrypted.
14–15	Reserved.

The following is an example of coding for a supervisor call block for a Get Decrypted Value SVC:

```
        EVEN                        DECRYPT DATA IN BUFFER MSG2
GDV     BYTE >46
        BYTE 0
        TEXT '34544658'
        DATA 42
        DATA MGS2
        DATA 0
```

10.4 JOB ACCOUNTING

Job accounting support provided by DNOS consists of accumulating data related to the use of system resources by the job. The user must provide a program (or programs) to access this data and process it to provide reports, billings, files, or other required documents.

Job accounting software is an option that is selected when the system is generated. The ACCOUNTING group, consisting of two SVCs, must be included to implement the job accounting support.

Accounting information for all jobs is written to the job accounting file in chronological order. The program supplied by the user must sort the information by job identifier to obtain the data pertinent to each job.

DNOS automatically writes records to the accounting file at the following times:

- When the system is initially loaded (IPL)
- When a job is started
- When a spooled output terminates
- When a task terminates
- When a job terminates

In addition to the records that are written automatically, the user program may execute a Log Accounting Entry SVC to write a record in the file. The entry may include up to 70 bytes consisting of whatever data the application may require.

The accounting records are written to files .S$ACT1 and .S$ACT2. Initial records are written to file .S$ACT1. When the file is full, DNOS begins writing records to .S$ACT2 and calls a program (task ID >54 on utilities program file .S$UTIL) to process the data in file .S$ACT1. When each file is full, the program is called and subsequent records are written to the other file. The user must provide the program on the system program file to process the file according to the requirements of the site.

Another job accounting SVC is available to user programs. This SVC returns the accounting information being accumulated by an executing task. The SVC returns the same types of information as are written to the accounting file when the task terminates. The *DNOS System Programmer's Guide* explains the structure of the accounting file and gives the details concerning how to build an accounting log processor.

10.4.1 Logging an Accounting Entry

When the accounting requirements of a site or application require accounting entries from user tasks, the task may assemble the data for an entry in a buffer and execute an SVC to process the entry. The Log Accounting Entry SVC (opcode >47) passes the user's entry to the system, which adds a header and writes the record in the accounting file.

The supervisor call block for the SVC is as follows:

SVC >47 -- LOG ACCOUNTING ENTRY ALIGN ON WORD BOUNDARY

DEC	HEX		
0	0	>47	<RETURN CODE>
2	2	BUFFER ADDRESS	
4	4	RESERVED	

2279722

The call block contains the following:

Byte	Contents
0	Opcode, >47.
1	Return code. DNOS returns a zero in this byte when the operation completes satisfactorily, or an error code when the operation completes in error.
2-3	Buffer address. The address of the buffer that contains the user log entry.
4-5	Reserved.

The first byte in the buffer that contains the user log entry must contain the number of bytes in the entry. The maximum number of bytes in the entry is 70. There are no other restrictions on the buffer contents.

The following is an example of coding for a supervisor call block for a Log Accounting Entry SVC:

```
        EVEN                LOG ACCOUNTING ENTRY IN BUFFER
LAE     BYTE >47            ACCBUF
        BYTE 0
        DATA ACCBUF
        DATA 0
```

10.4.2 Accessing Accounting Data

As a task is executing, information is being accumulated for the accounting file entry to be written when the task terminates. This information is available to the task by executing a Get Accounting Information SVC (opcode >49). To access information for a task other than the calling task, the calling task must supply the run-time ID of the task. The system returns the following:

- Task state code (Table 10-1)
- CPU execution time
- SVC count
- I/O byte count
- Maximum memory used

The supervisor call block for the SVC is as follows:

SVC >49 -- Get Accounting Information   Align on word boundary

Dec	Hex		
0	0	>49	<Return Code>
2	2	Task ID	<State>
4	4	<CPU Execution Time>	
6	6		
8	8	<Number of SVCs Issued>	
10	A		
12	C	<Number of I/O Bytes Transferred>	
14	E		
16	10	<Maximum Memory Used>	
18	12	Reserved	

2279723

The call block contains the following:

Byte	Contents
0	Opcode, >49.
1	Return code. DNOS returns a zero in this byte when the operation completes satisfactorily, or an error code when the operation completes in error.
2	Zero or task ID. Run-time ID of task for which accounting information is required. When zero is entered, DNOS returns accounting information for the calling task.
3	Task state code. Returned by the system. The state of the task in the form of one of the codes listed in the table of task state codes.
4–7	CPU execution time. Returned by the system. Number of clock ticks of use of the CPU by the task.
8–11	Number of SVCs issued. Returned by the system. The count of SVCs issued by the task.
12–15	Number of I/O bytes transferred. Returned by the system. The count of bytes transferred by I/O operations.
16–17	Maximum memory used. Returned by the system. The maximum number of bytes of user memory allocated to the task at one time.
18–19	Reserved.

The user task supplies zero or a task run-time ID in byte 2. Enter zero to access accounting information for the calling task. Enter the run-time ID to access accounting information for another task. The run-time ID is returned to the caller of the Execute Task SVC when a task is executed in this way. A task may obtain its run-time ID by executing a Self-Identification SVC.

The CPU execution time returned by the system is a count of clock ticks. On a computer that uses 60 Hz power, a clock tick is 8.33 ms. When the power line frequency is 50 Hz, a clock tick is 10 ms.

The following is an example of coding for a supervisor call block for a Get Accounting Information SVC:

```
        EVEN          GET ACCOUNTING INFORMATION FOR
GAI     BYTE >49      CALLING TASK AFTER MOVING
        BYTE 0        RUN-TIME ID INTO BYTE AT
TRID    BYTE 0        TRID
STSK    BYTE 0
TIME    DATA 0,0
SVCC    DATA 0,0
IOBC    DATA 0,0
MMEM    DATA 0
        DATA 0
```

10.5 MEMORY CONTROL

The architecture of the Model 990 computer provides an address space approaching 64K bytes for each task. The Model 990/10 Computer and the Model 990/12 Computer use hardware memory mapping to provide access to any area of available memory by mapping from one to three segments into a 64K-byte logical address space. Each segment consists of an integral number of beets. A beet is a block of memory consisting of 32 bytes; each beet begins at an address that is an even multiple of 32.

For example, a task may use a single segment containing the data and the executable code. In this case, the task is in a contiguous area of memory, and that area of memory is mapped into the task's address space.

Another task may use two segments of memory, one for the data, and the other for the executable code. The Link Editor defines the sizes and boundaries of the segments, and the system maps these segments into the address space of the task, regardless of the actual addresses in memory of each segment.

An example of a three-segment task is one in which the data occupies a segment, part of the executable code occupies another segment, and the remainder of the executable code occupies the third segment. The sizes and boundaries of the segments are supplied by the Link Editor according to the needs of the task; the only restriction is that segment boundaries must be beet addresses (multiples of 32 bytes). The system sets the mapping registers to associate the actual addresses in which the code resides with the corresponding addresses in the task's memory area.

DNOS provides supervisor calls that allow the task to alter the size of its memory area. The task may execute a Get Memory SVC to obtain memory for buffers required during execution of the task. When the buffers are no longer required, the task may execute a Release Memory SVC to release this memory. In addition, the Load Overlay SVC can be used to load an overlay into the memory space of the task.

DNOS also provides the Segment Management SVC. This SVC allows the task to dynamically change the current set of segments for the task, to guarantee access to a segment by the task, and to write segments to the system disk.

10.5.1 Requesting Memory

When a task requires more memory (for a buffer, or for dynamic tables, for example), the task can issue an SVC to increase the size of the memory allocated to the task. The Get Memory SVC (opcode >12) allocates a specified number of beets of memory to the calling task, and returns the beginning address of the first beet in workspace register 9. The calling task may not be a memory-resident task, nor may it have intertask common memory mapped into its address space. (For details of intertask common memory, see the Get Common Data Address SVC.) The segment of the task that is being expanded must be the last segment mapped into the task. The total available address space is >7FF beets. The sum of the number of beets allocated to the task plus the requested number of beets must be less than >800.

When DNOS executes the SVC, it swaps the task out of memory, then swaps it again into an expanded area of memory large enough for the original size of the task plus the requested area. DNOS also modifies the mapping registers appropriately. A task may execute additional Get Memory SVCs as required, until the size of the task reaches the maximum memory size.

The supervisor call block for the SVC is as follows:

SVC >12 -- GET MEMORY ALIGN ON WORD BOUNDARY

DEC	HEX		
0	0	>12	<RETURN CODE>
2	2	BEETS	

2279724

The call block contains the following:

Byte	Contents
0	Opcode, >12.
1	Return code. DNOS returns zero when the operation completes satisfactorily. When the operation completes in error, DNOS returns an error code.
2-3	Number of beets of memory to be allocated.

The following is an example of coding for a supervisor call block for a Get Memory SVC:

```
        EVEN                ALLOCATE 16 ADDITIONAL BEETS OF
SCBB    BYTE >12            MEMORY AND RETURN ADDRESS IN R9.
        BYTE 0
BCNT    DATA 16
```

10.5.2 Releasing Memory

When a task has no further need of memory, the task can issue an SVC to release memory. The SVC can release memory allocated to the task when it was loaded, or memory allocated by the Get Memory SVC call. The Release Memory SVC (opcode >13) releases all memory beyond the address in workspace register 9. The calling task may not be a memory resident task, and it may not have intertask common memory mapped into its address space. Also, any initiate mode I/O requested by the calling task must have completed before issuing a Release Memory SVC. DNOS modifies the mapping registers of the task to address only the remaining memory.

The supervisor call block for the SVC is as follows:

SVC >13 -- RELEASE MEMORY ALIGN ON WORD BOUNDARY

DEC	HEX		
0	0	>13	<RETURN CODE>
2	2	[RESERVED]	

2279725

The call block contains the following:

Byte	Contents
0	Opcode, >13.
1	Return code. DNOS returns zero when the operation completes satisfactorily. When the operation completes in error, DNOS returns an error code.
2-3	[Reserved].

The following is an example of coding for a supervisor call block for a Release Memory SVC:

```
         EVEN                      RELEASE MEMORY FROM ADDRESS IN R9
SCBC     BYTE >13                  THROUGH END OF TASK MEMORY.
         DATA 0
```

An example of coding to place an address in workspace register 9 and execute a Release Memory SVC is as follows:

```
         DXOP SVC,15               DEFINE XOP FOR SVC
           .
           .
           .
         LI   R9,>8400             RELEASE MEMORY FROM ADDRESS
         SVC  @SCBC                >8400 THROUGH END OF TASK.
```

10.5.3 Loading an Overlay

A task can load an overlay into the memory space of the task by executing a Load Overlay SVC (opcode >14). The task must provide space for the overlay and also space for the relocation bit map when relocation is required. Whether or not the overlay requires relocation is specified when the overlay is installed.

The size of the relocation bit map may be computed using the following formula:

$$SBM = 2 \times \left[\frac{SOV + 31}{32} \right]$$

SOV is the size of the overlay in bytes. The size of the bit map (SBM, also in bytes) is rounded up to the next higher integer.

The overlay must be loaded at the natural load address (the address provided by the Link Editor when the task is linked) unless relocation is not required or the overlay has been installed relocatable. In those cases, the overlay is loaded at any valid logical address.

The supervisor call block for the SVC is as follows:

SVC >14 -- LOAD OVERLAY ALIGN ON WORD BOUNDARY

DEC	HEX		
0	0	>14	<RETURN CODE>
2	2	LOAD ADDRESS	
4	4	OVERLAY ID	
6	6	PROGRAM FILE LUNO	

2279447

The call block contains the following:

Byte **Contents**

0 Opcode, >14.

1 Return code. DNOS returns zero when the operation completes satisfactorily. When the operation completes in error, DNOS returns an error code.

2-3 Load address, or zero. When this field contains zero, the overlay is loaded at the natural load address. Otherwise, the overlay is loaded at the address in this field. When the overlay was installed as relocatable and the address in this field is not the natural load address, relocation is performed.

4-5 Overlay ID. The overlay ID under which the overlay was installed.

6 Program file LUNO, zero, or >FF. When this field contains zero, the overlay is loaded from S$SHARED. When this field contains >FF, the overlay is loaded from the program file from which the calling task was loaded.

The following is an example of coding for a supervisor call block for a Load Overlay SVC:

```
        EVEN          LOAD OVERLAY >6A AT NATURAL LOAD
LDOVLY  BYTE >14      ADDRESS. LOAD FROM PROGRAM FILE
LDOERR  BYTE 0        LUNO >1A
        DATA 0
        DATA >6A
        BYTE >1A
```

10.5.4 Managing Memory Segments

In addition to the segments previously defined for a program, additional disk- or memory-based segments may be provided. These segments may be dynamically mapped into or removed from the memory area of the task.

Disk-based segments are installed on program files using either the Install Task Segment (IT), Install Procedure Segment (IP), or Install Program Segment (IPS) commands, or the Install Task SVC or the Install Procedure/Program Segment SVC. Disk-based segments are loaded into memory and mapped into memory areas of tasks by the segment manager. The segment manager maintains a count of the tasks that require the segment, and disposes of segments that are no longer required. When the segment contents have not changed, or when changes to the disk copy are not permitted, segment manager releases the memory space occupied by the segment. Otherwise, segment manager writes the segment to the program file and releases the memory space.

A task may request the segment manager to hold a segment in memory even when no task currently requires the segment. This allows passing data to another task that may not be currently executing. When the segment is no longer required, a task may request the segment manager to release the segment.

DNOS includes a structure for retaining in physical memory recently used disk-based segments that are no longer required by any executing task. This structure is referred to as the *software cache list*. This approach presupposes that a recently used segment is likely to be reused soon.

A task can have one, two, or three map segments. The identity of the *last segment* depends on the linking and execution of the task. Each procedure segment, program segment, or task segment occupies one map segment position. Procedure segments always occupy map segment positions that precede the position occupied by the task segment. If program segments are used, they usually occupy map segment positions that follow the position occupied by the task segment. Program segments are dynamically mapped into the task segment or added during execution.

Memory-based segments are created by the segment manager as uninitialized segments, and are mapped into the memory area of the task that requests their creation. The task then writes the required data into the segment. Shared memory-based segments are a means of passing data between tasks in the same job or in different jobs. A task must reserve this type of segment or the memory is released when the segment is released.

Attributes are defined for a segment when it is installed. The attributes of a task segment are specified in the Install Task SVC that installs the task on a program file. Similarly, the attributes of a procedure segment or a disk-based program segment are specified in the Install Procedure/Program Segment SVC. The attributes of a memory-based segment are specified when the segment is created by the segment manager. The segment attributes are as follows:

- Readable — Segment may be accessed in memory for read operations.
- System — Segment may only be accessed by system tasks.
- Memory resident — Segment remains accessible in memory.
- Replicatable — More than one copy may exist in memory.

- Share protected — Segment may not be shared concurrently by two or more tasks.
- Writable control store — Segment contains executable code that accesses writable control store.
- Execute protected — Segment contents may not be executed.
- Write-protected — Segment contents may not be altered in memory.
- Updatable — Segment will be written to its permanent file position on disk if it has been marked modified.
- Reusable — Segment may be used consecutively without reloading. This segment may reside on the software cache list while memory space is available for it.
- Copyable — Segment may be replicated by copying the segment from the memory copy.

The following attributes apply only to task segments:

- Privileged — Segment has been installed in a program file as a hardware privileged task segment. The task can execute hardware privileged instructions.
- Software privileged — Segment has been installed in a program file as a software privileged task segment. The task can issue privileged supervisor calls.
- Overflow protected — Segment has been installed in a program file with overflow protection. During execution, arithmetic overflow is detected as a fatal task error.

A user program calls the Segment Management SVC (opcode >40) to request the services of the segment manager. The following operations may be requested by a task:

- Change Segment
- Create Segment
- Reserve Segment
- Release Segment
- Check Segment Status
- Force Write Segment
- Set or Reset Not Modified and Releasable Flag
- Load Segment
- Unload Segment

- Set Exclusive Use of Segment
- Reset Exclusive Use of Segment

The Change Segment operation is not valid for the task segment. With that exception, the segment operations may be performed on any segment.

The supervisor call block for the SVC is as follows:

SVC >40 -- Segment Management — Align on word boundary

Dec	Hex		
0	0	>40	<Return Code>
2	2	Sub-Opcode	Segment Group LUNO
4	4	Flags	
6	6	Segment ID One (Installed or Run-time ID)	
8	8		
10	A	Segment ID Two (Run-time ID)	
12	C	<Segment Address in Address Space>	
14	E	<Segment Length>	
16	10	Attributes	
18	12	[Reserved]	

2279726

The call block contains the following:

Byte	Contents
0	Opcode, >40.
1	Return code. DNOS returns zero when the operation completes satisfactorily. When the operation completes in error, DNOS returns an error code in this byte.
2	Sub-opcode: 00 — Change Segment. 01 — Create Segment. 02 — Reserve Segment. 03 — Release Segment. 04 — Check Segment Status. 05 — Force Write Segment. 06 — Reserved. 07 — Set/Reset Not Modified and Releasable. 08 — Reserved. 09 — Load Segment. 0A — Unload Segment. 0B — Set Exclusive Use of Segment. 0C — Reset Exclusive Use of Segment. 0D — Reserved.
3	Segment group LUNO. The LUNO assigned to the segment group (program file) that contains the segment. Does not apply to memory-based segments. Enter zero when the program file is S$SHARED, or >FF when the program file is the file on which the calling task resides.
4–5	Flags. See descriptions of each sub-operation for details.
6–9	Segment ID one. When bit zero of the flags word is set to 1, the installed ID of a segment. Otherwise, nothing or the run-time ID of a segment. Segment ID must be right justified in the field.
10–11	Segment ID two. Run-time ID of a segment. DNOS returns a segment run-time ID in this field.
12–13	Segment address, returned by the system. The logical address within the task of the first byte of the segment.
14–15	Segment length, supplied by the user for the Create Segment operation. Returned by the system for other operations. The number of bytes in the segment.
16–17	Attributes. Used to define attributes for a Create Segment operation. Returned by the system for a Check Segment Status operation. See descriptions of each suboperation for details.
18–19	[Reserved].

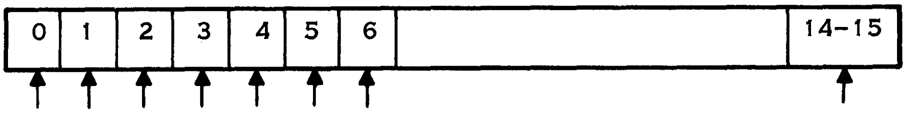
10.5.4.1 Changing Segments. A task can add a segment, exchange segments, or delete a segment by executing an SVC. When a segment is being added, the segment must be added in the position after the last one currently in use (last segment) and must not be larger than the remaining portion of the task memory area. When exchanging segments other than the last segment, for non-system tasks the new segment must be the same size as the old segment. When exchanging the last segment, the new segment need not be the same size as the old segment, but it must not be larger than the portion of the task memory area remaining after the old segment is removed. The task segment may not be exchanged. When deleting, the segment must be in the last position currently in use and may not be the task segment.

To retain the segment memory when the segment is released from the task's address space, a reserve segment operation must have been performed prior to exchanging a segment.

The Change Segment operation (sub-opcode >00) adds a segment, exchanges segments, or deletes a segment of the calling task. The following fields of the supervisor call block apply to the operation:

- Opcode, >40
- Return code
- Sub-opcode, >00
- Segment group LUNO
- Flags
- Segment ID One
- Segment ID Two
- Segment address
- Segment length
- Attributes

The flags used by the Change Segment operation are as follows:



2279727

Bit 0 — Installed ID flag. Set as follows:
1 — The new segment is specified by installed ID in Segment ID One, bytes 6-9.
0 — The new segment is specified by run-time ID in Segment ID One, bytes 6-9.

Bit 1 — Not modified flag. Set as follows:
1 — Old segment has not been modified.
0 — Old segment has been modified.

Bit 2 — Releasable flag. Set as follows:
1 — Old segment will be released from memory when it is no longer in use by tasks.
0 — Old segment may be placed on the software cache list when it is no longer in use by tasks.

Bit 3 — Memory-based segment flag. Set as follows:
1 — New segment is memory-based. LUNO in byte 3 is ignored.
0 — New segment is disk-based. LUNO in byte 3 must be assigned to the file of the segment.

Bit 4 — Run-time ID flag. Set as follows:
1 — Old segment is specified by run-time ID in Segment ID Two, bytes 10-11.
0 — Old segment is specified by position number, bits 14-15.

Bit 5 — Task segment flag. Set as follows:
1 — New segment is a task segment.
0 — New segment is not a task segment.

Bit 6 — Verify load address. Set as follows:
1 — Verify the logical address within the task of the changed segment is the same as the load point of the segment on the program file. If this does not verify, an error code is returned and the segment is not changed.
0 — Do not verify the load address.

Bit 7 — Set or reset exclusive use enable flag. Set as follows:
1 — The set/reset exclusive use flag applies.
0 — The set/reset exclusive use flag does not apply.

Bit 8 — Set or reset exclusive use flag. Set as follows:
1 — Set exclusive use for the old segment.
0 — Reset exclusive use for the old segment.

Bits 14-15 — Position number. When bit 4 is set to zero, these bits must contain a nonzero value, as follows:
01 — Old segment is segment 1.
10 — Old segment is segment 2.
11 — Old segment is segment 3.

If the set or reset exclusive use enable flag (bit 7) is set to one, the function indicated by the set/reset exclusive use flag (bit 8) is applied to the old segment. Thus, if bit 8 is set to one, exclusive use is set for the old segment; and if bit 8 is set to zero, exclusive use is reset for the old segment. The set or reset function is applied before the Change Segment operation, so that if the set or reset fails, the Change Segment operation is not attempted. If the set or reset succeeds but the Change Segment operation fails, the result of the set or reset is not changed.

The LUNO specified applies to the new segment. The ID of the new segment is specified in the Segment ID One field. If the ID of the old segment is supplied, it is supplied in the Segment ID Two field.

To use the Change Segment operation to add a segment, use one of these methods:

- Specify a run-time ID of >FFFF for the old segment.
- Specify a position number for the old segment. This position must be one greater than the last position currently in use.

A segment can be added only when fewer than three segments are in use. Either the installed or run-time ID of the new segment must be supplied.

To use the Change Segment operation to exchange segments, use the following guidelines:

- Specify either the installed ID or the run-time ID of the new segment.
- Specify either the run-time ID or position of the old segment.
- If the old segment is specified by run-time ID, the new segment position is that previously occupied by the old segment.

Only the segment in the last position currently in use may be deleted by a Change Segment operation. To delete a segment, use the following guidelines:

- Specify either the run-time ID or position of the old segment.
- Set the memory-based flag to one.
- Specify a run-time ID of >FFFFFFFF for the new segment.

If the Change Segment operation completes without error, the segment manager returns the run-time ID of the new segment in the Segment ID Two field of the supervisor call block. The segment manager returns the logical address of the first byte of the segment in the segment address field; it returns the number of bytes in the segment in the segment length field. For a Change Segment operation that deletes a segment, only the Segment ID Two field is altered. It contains the value >FFFF.

The Change Segment operation can also map a physical record from a relative record file into a task. To do this, specify a LUNO that is assigned to the desired file and opened by the requesting task. Specify the record number of the desired physical record (not logical record) as an installed ID in Segment ID One. In the attributes field, specify the desired values for bits 5 and 10; all other bits are ignored. (The attributes field is not used if the LUNO is assigned to a program file.) The meanings of bits 5 and 10 of the attributes field are as follows:

Bit 5 — Share protected. Assign as follows:
1 — Assign share protected attribute to new segment.
0 — Do not assign share protected attribute to new segment.

Bit 10 — Execute protected. Assign as follows:
1 — Assign execute protected attribute to new segment.
0 — Do not assign execute protected attribute to new segment.

The following is an example of coding for a supervisor call block for a Change Segment operation:

```
         EVEN              PLACE SEGMENT >5C ON PROGRAM
CHSEG    BYTE >40          FILE LUNO >3A IN POSITION 3.
         BYTE 0            OLD SEGMENT HAS NOT BEEN MODIFIED
         BYTE 0            AND IS RELEASABLE.
         BYTE >3A
         DATA >E003
         DATA 0,>5C
SRID     DATA 0
SADR     DATA 0
SLEN     DATA 0
         DATA 0,0
```

10.5.4.2 Creating Segments. A task can create a memory-based program segment by executing an SVC. When the created program segment is added to the segment set of the calling task, the program segment must be added as the last segment, and must not be larger than the remaining portion of the task memory area. For user tasks, when the program segment is exchanged for a segment other than the last segment, the newly created program segment must be the same size as the old segment. For a system task, the newly created program segment may be larger or smaller than the segment it replaces.

For any task, when the program segment is exchanged for the last segment, the newly created program segment need not be the same size as the old segment, but it must not be larger than the portion of the task memory area remaining after the old segment is removed.

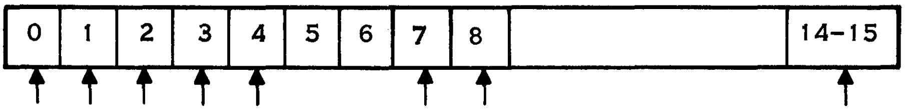
To retain the segment memory when the segment is released from the task's address space, a reserve segment operation must be performed prior to exchanging a segment.

The Create Segment operation (sub-opcode >01) creates a program segment and exchanges it for a segment of the calling task, or adds it to the calling task. The following fields of the supervisor call block apply to the operation:

- Opcode, >40
- Return code

- Sub-opcode, >01
- Flags
- Segment ID Two
- Segment address
- Segment length
- Attributes

The flags used by the Create Segment operation are as follows:



2279728

Bit 0 — Installed ID flag. Set as follows:
1 — The new segment is specified by installed ID in Segment ID One, bytes 6-9.
0 — The new segment is specified by run-time ID in Segment ID One, bytes 6-9.

Bit 1 — Not modified flag. Set as follows:
1 — Old segment has not been modified.
0 — Old segment has been modified.

Bit 2 — Releasable flag. Set as follows:
1 — Old segment will be released from memory when it is no longer in use by tasks.
0 — Old segment may be placed on the software cache list when it is no longer in use by tasks.

Bit 3 — Memory-based segment flag. Set as follows:
1 — New segment is memory-based. LUNO in byte 3 is ignored.
0 — New segment is from a relative record file. LUNO in byte 3 must be assigned to the file of the segment.

Bit 4 — Run-time ID flag. Set as follows:
1 — Old segment is specified by run-time ID in Segment ID Two, bytes 10–11.
0 — Old segment is specified by position number, bits 14–15.

Bit 7 — Set or reset exclusive use enable flag. Set as follows:
1 — The set/reset exclusive use flag applies.
0 — The set/reset exclusive use flag does not apply.

Bit 8 — Set or reset exclusive use flag. Set as follows:
1 — Set exclusive use for the old segment.
0 — Reset exclusive use for the old segment.

Bits 14–15 — Position number. When bit 4 is set to zero, these bits must contain a nonzero value, as follows:
01 — Old segment is segment 1.
10 — Old segment is segment 2.
11 — Old segment is segment 3.

If the set or reset exclusive use enable flag (bit 7) is set to one, the function indicated by the set/reset exclusive use flag (bit 8) is applied to the old segment. Thus, if bit 8 is set to one, exclusive use is set for the old segment; and if bit 8 is set to zero, exclusive use is reset for the old segment. The set or reset function is applied before the Create Segment operation, so that if the set or reset fails, the Create Segment operation is not attempted. If the set or reset succeeds but the Create Segment operation fails, the result of the set or reset is not changed.

When using the Create Segment operation to add a segment, use one of these methods:

- Specify a run-time ID of >FFFF for the old segment in the Segment ID Two field.

- Specify a position number for the old segment. This position must be one greater than the last position currently in use.

When using the Create Segment operation to exchange segments, use one of these methods:

- Specify the run-time ID of the old segment to be replaced in the Segment ID Two field.

- Specify the position of the old segment to be replaced.

Specify the desired length of the new segment, in bytes, in the segment length field.

The following bits in the attributes field (bytes 16–17) may be set to one to assign the corresponding attributes:

Bit 0 — Readable. Set as follows:
1 — Assign readable attribute to new segment.
0 — Do not assign readable attribute to new segment.

Bit 1 — System. Set as follows:
1 — New segment is a system segment.
0 — New segment is not a system segment.

Bit 5 — Share protected. Set as follows:
1 — Assign share protected attribute to new segment.
0 — Do not assign share protected attribute to new segment.

Bit 9 — Writable control store. Set as follows:
1 — Segment uses writable control store.
0 — Segment does not use writable control store.

Bit 10 — Execute protected. Set as follows:
1 — Assign execute protected attribute to new segment.
0 — Do not assign execute protected attribute to new segment.

Bit 11 — Write protected. Set as follows:
1 — Assign write protected attribute to new segment.
0 — Do not assign write protected attribute to new segment.

Bit 13 — Reusable. Set as follows:
1 — Assign reusable attribute to new segment.
0 — Do not assign reusable attribute to new segment.

If the operation completes without error, the segment manager returns the run-time ID of the new segment in the Segment ID Two field and the mapped address of the first byte of the segment in the segment address field.

The Create Segment operation is also used to map a physical record from a relative record file into a task. To do this, specify the following:

- A LUNO that is both assigned to the desired file and opened by the requesting task.

- The record number of the desired physical record (not logical record) as an installed ID in Segment ID One.

- The desired values for bits 5 and 10 in the attributes field. All other bits are ignored.

- The physical record length of the file to be the same as the segment length.

If the requested segment corresponds to an existing segment, an error is returned and the segment is not changed.

The following is an example of coding for a supervisor call block for a Create Segment operation:

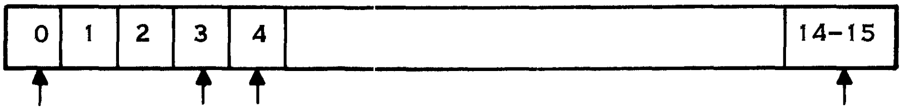
```
        EVEN            CREATE MEMORY-BASED SEGMENT HAVING
GES     BYTE >40        SYSTEM, EXECUTE PROTECTED, AND
        BYTE 0          WRITE PROTECTED ATTRIBUTES. OLD
        BYTE >01        SEGMENT, RUN-TIME ID 47, HAS NOT
        BYTE 0          BEEN MODIFIED AND IS RELEASABLE.
        DATA >7800      THE SEGMENT IS TO BE 1000 BYTES.
        DATA 0,0
SRID    DATA >47
SADR    DATA 0
        DATA 1000
        DATA >4030
        DATA 0
```

10.5.4.3 Reserving Segments. A task can reserve a segment by executing an SVC. This operation is used to keep a memory-based or disk-based segment from being destroyed when not in use. A segment is reserved by entering the segment in the reserved segment list of the current job. The operation applies to disk-based segments and memory-based segments. A segment remains reserved until it is removed from the reserved list by a Release Segment operation, or until the current job is terminated. The segment continues to be reserved after the job terminates if it is on the reserved list of another job.

The Reserve Segment operation (sub-opcode >02) reserves the specified segment. The following fields of the supervisor call block apply to the operation:

- Opcode, >40
- Return code
- Sub-opcode, >02
- Segment group LUNO
- Flags
- Segment ID One

The flags used by the Reserve Segment operation are as follows:



2279729

Bit 0 — Must be set to zero.

Bit 3 — Memory-based segment flag. Set as follows:
1— Memory-based segment. LUNO in byte 3 is ignored.
0 — Disk-based segment.

Bit 4 — Run-time ID flag. Set as follows:
1 — Segment is specified by run-time ID in Segment ID One, bytes 6-9.
0 — Segment is specified by position number, bits 14-15.

Bits 14–15 — Position number. When bit 4 is set to zero, these bits must contain a nonzero value, as follows:
01 — Segment is segment 1.
10 — Segment is segment 2.
11 — Segment is segment 3.

The segment group LUNO must contain the LUNO assigned to the program file that contains the segment unless bit 3 is set to one.

When the run-time ID flag is set to one, the Segment ID One field contains the run-time ID of the segment to be reserved. Otherwise, the segment specified in the position number field is reserved.

The following is an example of coding for a supervisor call block for a Reserve Segment operation:

```
        EVEN                 RESERVE SEGMENT IN POSITION 2
RSS     BYTE >40             PROGRAM FILE LUNO IS >3F
        BYTE 0
        BYTE >02
        BYTE >3F
        DATA >0002
        DATA 0,0
        DATA 0
        DATA 0
        DATA 0
        DATA 0
        DATA 0
```

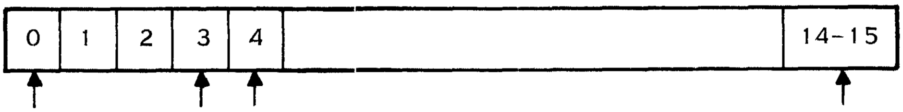
10.5.4.4 Releasing Reserved Segments. A task can release a reserved segment by executing an SVC. A segment is released by removing the segment from the reserved segment list of the current job. The operation applies to disk-based and memory-based segments that have been reserved; an error is returned for any segment not on the for reserved list for the current job.

After removing the segment from the reserved list, the segment manager checks to determine whether or not the segment is required in memory. If a segment is not memory-based, is not releasable, and is reusable, it will remain in memory if space is available. Otherwise, it is deleted.

The Release Segment operation (sub-opcode >03) releases the specified segment. The following fields of the supervisor call block apply to the operation:

- Opcode, >40
- Return code
- Sub-opcode, >03
- Segment group LUNO
- Flags
- Segment ID One

The flags used by the Release Segment operation are as follows:



2279730

Bit 0 — Must be set to zero.

Bit 3 — Memory-based segment flag. Set as follows:
1 — Memory-based segment. LUNO in byte 3 is ignored.
0 — Disk-based segment.

Bit 4 — Run-time ID flag. Set as follows:
1 — Segment is specified by run-time ID in Segment ID One, bytes 6-9.
0 — Segment is specified by position number, bits 14-15.

Bits 14-15 — Position number. When bit 4 is set to zero, these bits must contain a nonzero value, as follows:
01 — Segment is segment 1.
10 — Segment is segment 2.
11 — Segment is segment 3.

The segment group LUNO must contain the LUNO assigned to the program file that contains the segment, unless bit 3 is set to 1.

When the run-time ID flag is set to one, the Segment !D One field contains the run-time ID of the segment to be released. Otherwise, the segment in the position specified is released.

The following is an example of coding for a supervisor call block for a Release Segment operation:

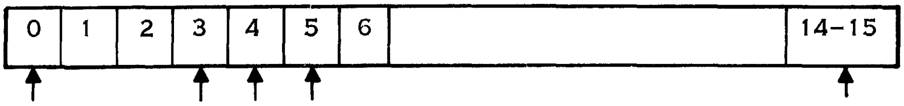
```
       EVEN              RELEASE SEGMENT WITH RUN-TIME ID >34
RLS    BYTE >40          ON PROGRAM FILE LUNO >3F
       BYTE 0
       BYTE >03
       BYTE >3F
       DATA >0800
       DATA 0,>34
       DATA 0
       DATA 0
       DATA 0
       DATA 0
       DATA 0
```

10.5.4.5 Checking Segment Status. A task can obtain the status of a segment by executing an SVC. The segment must either be in memory, or be swapped from memory to disk temporarily. The calling task supplies either an installed or run-time ID or a position in the task address space for the segment. Segment manager returns the state of the task segment flag, the segment length, and the attributes. For a segment that is mapped into the memory area of the calling task, segment manager also returns the logical address of the first byte of the segment. The run-time ID is returned in the Segment ID Two field. The installed ID is returned in the Segment ID One field.

The Check Segment Status operation (sub-opcode >04) returns the status of the specified segment. The following fields of the supervisor call block apply to the operation:

- Opcode, >40
- Return code
- Sub-opcode, >04
- Segment group LUNO
- Flags
- Segment ID One
- Segment ID Two
- Segment address
- Segment length
- Attributes

The flags used by the Check Segment Status operation are as follows:



2279731

Bit 0 — Installed ID flag. Set as follows:
1 — The segment is specified by installed ID in Segment ID One, bytes 6-9.
0 — The run-time ID flag applies.

Bit 3 — Memory-based segment flag. Set as follows:
1 — Memory-based segment. LUNO in byte 3 is ignored.
0 — Disk-based segment.

Bit 4 — Run-time ID flag. Set as follows:
1 — Segment is specified by run-time ID in Segment ID One, bytes 6-9.
0 — Segment is specified by position number, bits 14–15.

Bit 5 — Task segment flag. Set as follows:
1 — Segment is a task segment.
0 — Segment is not a task segment.

Bits 14–15 — Position number. When bits 0 and 4 are set to zero, these bits must contain nonzero value, as follows:
01 — Segment is segment 1.
10 — Segment is segment 2.
11 — Segment is segment 3.

When the memory-based segment flag is set to zero, the segment group LUNO must contain the LUNO assigned to the program file that contains the segment for which status is being checked. When the flag is set to one, the field is ignored.

When the installed ID flag is set to one, the Segment ID One field contains the installed ID of the segment for which status is being checked. When the installed ID flag is set to zero and the run-time ID flag is set to one, the field contains the run-time ID. In all cases, the installed ID is returned right-justified in the field.

If the installed ID flag is set to zero, the memory-based segment flag should be set to indicate if the segment for which status is sought is a memory-based segment. This flag is set to one if the segment is memory-based and set to zero for disk segments.

If the installed ID flag is set to one, the task segment flag should be set to indicate if the segment for which status is sought is a task segment. This flag is set to one if the segment is installed as a task in a program file and set to zero otherwise.

When the run-time ID flag is set to zero, and the position number is set in bits 14–15, the status of the segment in the specified position is returned.

When the specified segment is mapped into the memory address space of the calling task, the segment manager returns the mapped address of the first byte of the segment in the segment address field.

The segment manager returns the number of bytes in the segment in the segment length field.

The attributes are returned in the attributes field. The bits in the field have a different significance for task segments and for other segments. The attribute meanings are as follows:

Bit 0 — Readable. Set as follows:
1 — Segment may be read.
0 — Segment may not be read.
— For a task segment:
1 — Privileged.
0 — Not privileged.

Bit 1 — System. Set as follows:
1 — Segment is a system segment.
0 — Segment is not a system segment.

Bit 2 — Memory-resident. Set as follows:
1 — Segment is memory-resident.
0 — Disk-resident segment.

Bit 3 — Reserved.

Bit 4 — Replicatable. Set as follows:
1 — Segment is replicatable.
0 — Segment is not replicatable.

Bit 5 — Share protected. Set as follows:
1 — Segment is share protected.
0 — Segment is not share protected.
—For a task segment:
1 — Procedure 1 on S$SHARED.
0 — Procedure 1 not on S$SHARED.

Bit 6 — For a task segment:
1 — Procedure 2 on S$SHARED.
0 — Procedure 2 not on S$SHARED.

Bit 7 — Reserved.

Bit 8 — For a task segment:
1 — Overflow protection is enabled.
0 — Overflow protection is not enabled.

Bit 9 — Writable control store. Set as follows:
1 — Segment uses writable control store.
0 — Segment does not use writable control store.

Bit 10 — Execute protected. Set as follows:
1 — Segment is execute protected.
0 — Segment is not execute protected.

Bit 11 — Write protected. Set as follows:
1 — Segment is write protected.
0 — Segment is not write protected.
— For a task segment:
1 — Task is software privileged.
0 — Task is not software privileged.

Bit 12 — Updatable. Set as follows:
1 — Segment is updatable.
0 — Segment is not updatable.

Bit 13 — Reusable. Set as follows:
1 — Segment is reusable.
0 — Segment is not reusable.

Bit 14 — Copyable. Set as follows:
1 — Segment is copyable.
0 — Segment is not copyable.

Bit 15 — For a task segment, use:
1 — Task can bypass file security
0 — Task can not bypass file security

The following is an example of coding for a supervisor call block for a Check Segment Status operation:

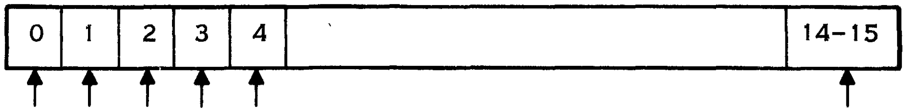
```
         EVEN            CHECK STATUS OF RUN-TIME SEGMENT ID
CSS      BYTE >40        >34 ON PROGRAM FILE LUNO >3F
         BYTE 0          WHICH IS NOT A TASK SEGMENT
         BYTE >04
         BYTE >3F
TFLG     DATA >0800
         DATA 0
RTID     DATA >34
         DATA 0
ADR      DATA 0
LEN      DATA 0
ATTR     DATA 0
         DATA 0
```

10.5.4.6 Force Writing Segments. A disk-based segment that has the updatable attribute and has been modified is copied to disk after it is no longer required in memory. A task may force that segment to be written immediately by executing an SVC. If the segment is not reserved and is not being used in a task and if the releasable flag is set to one, the segment is released from memory.

The Force Write Segment operation (sub-opcode >05) forces the specified segment to be written. The following fields of the supervisor call block apply to the operation:

- Opcode, >40
- Return code
- Sub-opcode, >05
- Segment group LUNO
- Flags
- Segment ID One

The flags used by the Force Write Segment operation are as follows:



2279732

Bit 0 — Installed ID flag. Set as follows:
1 — The segment is specified by installed ID in Segment ID One, bytes 6-9.
0 — The run-time ID flag applies.

Bit 1 — Not modified flag. Set as follows:
1 — Segment has not been modified.
0 — Segment has been modified.

Bit 2 — Releasable flag. Set as follows:
1 — Segment will be released from memory when it is no longer in use by tasks.
0 — Segment may be placed on the software cache list when it is no longer in use by tasks.

Bit 3 — Must be set to zero.

Bit 4 — Run-time ID flag. Set as follows:
1 — The segment is specified by run-time ID in Segment ID One, bytes 6-9.
0 — Segment is specified by position number, bits 14–15.

Bits 14-15 — Position number. When bits 0 and 4 are set to zero, these bits must contain a nonzero value, as follows:
01 — Segment is segment 1.
10 — Segment is segment 2.
11 — Segment is segment 3.

The segment group LUNO must contain the LUNO assigned to the program file that contains the segment to be written.

When the run-time ID flag is set to one, the Segment ID One field contains the run-time ID right-justified in the field.

The following is an example of coding for a supervisor call block for a Force Write Segment operation:

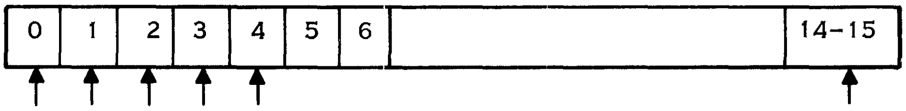
```
        EVEN                    WRITE INSTALLED PROCEDURE SEGMENT
FWS     BYTE >40                ID >68 ON PROGRAM FILE LUNO >3F
        BYTE 0
        BYTE >05
        BYTE >3F
        DATA >8000
        DATA 0
        DATA >68
        DATA 0
        DATA 0
        DATA 0
        DATA 0
        DATA 0
```

10.5.4.7 Setting and Resetting Segment Flags. The releasable flag and the not-modified flag of a segment are set and reset as part of the Change Segment, Create Segment, and Force Write Segment operations. The releasable flag is set to cause the segment manager to release the copy of the segment in memory. The not-modified flag should be reset by the user when the content of a segment is altered. This causes segment manager to copy the segment to disk if it is updatable. A task can set or reset these flags for a segment that is mapped into its memory address space by executing an SVC.

The Set/Reset Not Modified and Releasable operation (sub-opcode >07) sets the not-modified and releasable flags to the states specified in the call block. The following fields of the supervisor call block apply to the operation:

- Opcode, >40
- Return code
- Sub-opcode, >07
- Segment group LUNO
- Flags
- Segment ID One

The flags used by the Set/Reset Not Modified and Releasable operation are as follows:



2279733

Bit 0 — Must be set to zero.

Bit 1 — Not-modified flag. Set as follows:
1 — Segment has not been modified.
0 — Segment has been modified.

Bit 2 — Releasable flag. Set as follows:
1 — Segment will be released from memory when it is no longer in use by tasks.
0 — Segment may be placed on the software cache list when it is no longer in use by tasks.

Bit 3 — Must be set to zero.

Bit 4 — Run-time ID flag. Set as follows:
1 — The segment is specified by run-time ID in Segment ID One, bytes 6-9.
0 — Segment is specified by position number, bits 14-15.

Bits 14-15 — Position number. When bits 0 and 4 are set to zero, these bits must contain a nonzero value, as follows:
01 — Segment is segment 1.
10 — Segment is segment 2.
11 — Segment is segment 3.

The segment group LUNO must contain the LUNO assigned to the program file that contains the segment for which the flags are to be set or reset.

When the run-time ID flag is set to one, the Segment ID One field contains the run-time ID right-justified in the field.

The following is an example of coding for a supervisor call block for a Set/Reset Not Modified and Releasable operation:

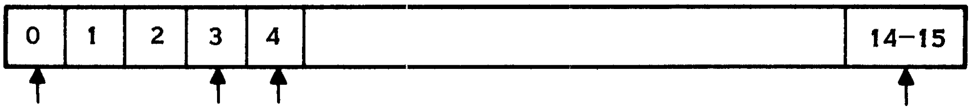
```
        EVEN                  SET RELEASABLE AND RESET NOT
SRMR    BYTE >40              MODIFIED FOR SEGMENT IN POSITION
        BYTE 0                2. PROGRAM FILE LUNO IS >3F
        BYTE >07
        BYTE >3F
        DATA >2002
        DATA 0,0
        DATA 0
        DATA 0
        DATA 0
        DATA 0
        DATA 0
```

10.5.4.8 Loading Segments. The Load Segment operation assures the user that the specified segment will be in memory while the task that issued the SVC is executing. The segment will not be mapped into the task address space. A segment can be loaded by more than one task, regardless of its attributes.

The Load Segment operation (sub-opcode >09) loads the specified segment. The following fields of the supervisor call block apply to the operation:

- Opcode, >40
- Return code
- Sub-opcode, >09
- Segment group LUNO
- Flags
- Segment ID One

The flags used by the Load Segment operation are as follows:



2283215

Bit 0 — Must be set to zero.

Bit 3 — Memory-based segment flag. Set as follows:
1 — Memory-based segment. LUNO in byte 3 is ignored.
0 — Disk-based segment.

Bit 4 — Run-time ID flag. Set as follows:
1 — Segment is specified by run-time ID in Segment ID One, bytes 6-9.
0 — Segment is specified by position number, bits 14-15.

Bits 14–15 — Position number. When bit 4 is set to zero, these bits must contain a nonzero value, as follows:
01 — Segment is segment 1.
10 — Segment is segment 2.
11 — Segment is segment 3.

The segment group LUNO must contain the LUNO assigned to the program file that contains the segment unless bit 3 is set to one.

When the run-time ID flag is set to one, the Segment ID One field contains the run-time ID of the segment to be loaded. Otherwise, the segment specified in the position number field is loaded.

The following is an example of coding for a supervisor call block for a Load Segment operation:

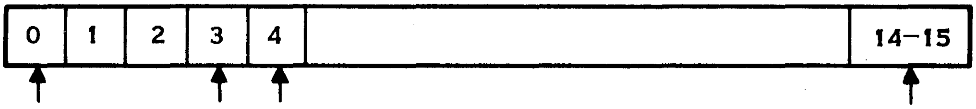
```
        EVEN                          LOAD SEGMENT WITH SEGMENT RUN ID
LDSM    BYTE >40                      OF >0204 PROGRAM FILE LUNO IS >3F
        BYTE 0
        BYTE >09
        BYTE >3F
        DATA >0800
        DATA 0
        DATA >0204
        DATA 0
        DATA 0
        DATA 0
        DATA 0
        DATA 0
```

10.5.4.9 Unloading Segments. The Unload Segment operation detaches the segment from the task so the segment does not need to be in memory when the task is in memory. An error is returned if the segment was not loaded by the task. If the reserve, use, and exclusive use counts are zero, the segment can be cached or deleted.

The Unload Segment operation (sub-opcode >0A) unloads the specified segment. The following fields of the supervisor call block apply to the operation:

- Opcode, >40
- Return code
- Sub-opcode, >0A
- Segment group LUNO
- Flags
- Segment ID One

The flags used by the Unload Segment operation are as follows:



2283216

Bit 0 — Must be set to zero.

Bit 3 — Memory-based segment flag. Set as follows:
1 — Memory-based segment. LUNO in byte 3 is ignored.
0 — Disk-based segment.

Bit 4 — Run-time ID flag. Set as follows:
1 — Segment is specified by run-time ID in Segment ID Two, bytes 6-9.
0 — Segment is specified by position number, bits 14-15.

Bits 14-15 — Position number. When bit 4 is set to zero, these bits must contain a nonzero value, as follows:
01 — Segment is segment 1.
10 — Segment is segment 2.
11 — Segment is segment 3.

The segment group LUNO must contain the LUNO assigned to the program file that contains the segment unless bit 3 is set to one.

When the run-time ID flag is set to one, the Segment ID One field contains the run-time ID of the segment to be reserved. Otherwise, the segment specified in the position number field is reserved.

The following is an example of coding for a supervisor call block for a Unload Segment operation:

```
        EVEN                   UNLOAD SEGMENT IN POSITION 2
ULSM    BYTE >40               PROGRAM FILE LUNO IS >3F
        BYTE 0
        BYTE >0A
        BYTE >3F
        DATA >0002
        DATA 0,0
        DATA 0
        DATA 0
        DATA 0
        DATA 0
        DATA 0
```

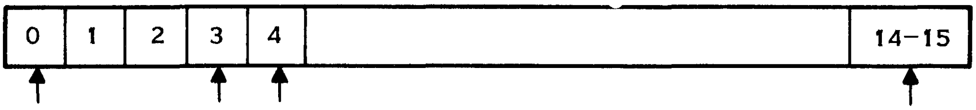
10.5.4.10 Setting Exclusive Use of Segments. The Set Exclusive Use of a Segment operation is used to extend the share protection attribute to segments not currently mapped in by a task. A segment that has exclusive use set is called an owned segment. Other users who try to map in the owned segment get a shared segment violation error unless the segment is replicatable, in which case a replicated copy will be mapped. The Set Exclusive Use operation also functions like a Reserve Segment operation, in that even if an owned segment has use and reserve counts of zero, the segment will not be deallocated.

If this SVC operation is to succeed, the following conditions must be met. The segment must not currently be owned by either the task issuing the SVC or another task. The segment must not be in use by any task other than the issuing task. Exclusive use of special table areas SMT, FMT, and PBM is not allowed.

The Set Exclusive Use of a Segment operation (sub-opcode >0B) sets exclusive use for the specified segment. The following fields of the supervisor call block apply to the operation:

- Opcode, >40
- Return code
- Sub-opcode, >0B
- Segment group LUNO
- Flags
- Segment ID One

The flags used by the Set Exclusive Use of a Segment operation are as follows:



2283216

Bit 0 — Must be set to zero.

Bit 3 — Memory-based segment flag. Set as follows:
1 — Memory-based segment. LUNO in byte 3 is ignored.
0 — Disk-based segment.

Bit 4 — Run-time ID flag. Set as follows:
1 — Segment is specified by run-time ID in Segment ID One, bytes 6-9.
0 — Segment is specified by position number, bits 14-15.

Bits 14–15 — Position number. When bit 4 is set to zero, these bits must contain a nonzero value, as follows:
01 — Segment is segment 1.
10 — Segment is segment 2.
11 — Segment is segment 3.

The segment group LUNO must contain the LUNO assigned to the program file that contains the segment unless bit 3 is set to one.

When the run-time ID flag is set to one, the Segment ID One field contains the run-time ID of the segment to be owned. Otherwise, the segment specified in the position number field is owned.

The following is an example of coding for a supervisor call block for a Set Exclusive Use of a Segment operation:

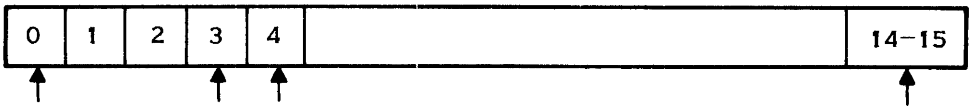
```
        EVEN                  SET EXCLUSIVE USE FOR MEMORY-BASED
SXSM    BYTE >40              SEGMENT WITH A SEGMENT RUN ID >0354
        BYTE 0
        BYTE >0B
        BYTE >0
        DATA >1800
        DATA 0,>0354
        DATA 0
        DATA 0
        DATA 0
        DATA 0
        DATA 0
```

10.5.4.11 Resetting Exclusive Use of Segments. The Reset Exclusive Use of a Segment operation relinquishes a task's ownership of a segment. The operation succeeds only if the segment is currently owned by the task issuing the SVC. The OSE is removed from the list of owned segments linked to the TSB and its memory released. If the segment is not in use or reserved, it is deleted.

The Reset Exclusive Use of a Segment operation (sub-opcode >0C) releases the specified segment. The following fields of the supervisor call block apply to the operation:

- Opcode, >40
- Return code
- Sub-opcode, >0C
- Segment group LUNO
- Flags
- Segment ID One

The flags used by the Reset Exclusive Use of Segment operation are as follows:



2283216

Bit 0 — Must be set to zero.

Bit 3 — Memory-based segment flag. Set as follows:
1 — Memory-based segment. LUNO in byte 3 is ignored.
0 — Disk-based segment.

Bit 4 — Run-time ID flag. Set as follows:
1 — Segment is specified by run-time ID in Segment ID One, bytes 6-9.
0 — Segment is specified by position number, bits 14-15.

Bits 14–15 — Position number. When bit 4 is set to zero, these bits must contain a nonzero value, as follows:
01 — Segment is segment 1.
10 — Segment is segment 2.
11 — Segment is segment 3.

The segment group LUNO must contain the LUNO assigned to the program file that contains the segment unless bit 3 is set to one.

When the run-time ID flag is set to one, the Segment ID One field contains the run-time ID of the segment to be reset. Otherwise, the segment specified in the position number field is reset.

The following is an example of coding for a supervisor call block for a Reset Exclusive Use of Segment operation:

```
        EVEN                RESET EXCLUSIVE USE FOR SEGMENT IN
RESM    BYTE >40            POSITION 2 — PROGRAM FILE LUNO IS >3F
        BYTE 0
        BYTE >0C
        BYTE >3F
        DATA >0002
        DATA 0,0
        DATA 0
        DATA 0
        DATA 0
        DATA 0
        DATA 0
```

10.6 TASK SYNCHRONIZATION

Task synchronization controls the execution of cooperating tasks to delay execution (or continuation of execution) of a task pending completion of all or a portion of another task. Synchronization also applies to interaction between a task and DNOS. DNOS supports the following types of task synchronization:

- Synchronization by messages
- Synchronization by semaphores
- Synchronization by events

Interprocess communication (IPC) transfers messages between tasks. The sending task issues a write request and the receiving task issues a read request. The actual transfer occurs when both requests have been issued. The transfer of the message synchronizes the tasks. The scope of task synchronization by messages is global. IPC is described in Section 8 of this manual.

Semaphores provide synchronization of tasks within a job. DNOS supports the use of semaphores with the Semaphore Operations SVC. The operations provided by this SVC can be combined in various ways to implement a variety of synchronization techniques.

Synchronization by events synchronizes a task with system functions. The system function is initiated as an event, and the task is suspended pending completion of the event. The Initiate Event SVC and the Wait for Event SVC provide event synchronization.

10.6.1 Using Semaphore Synchronization

A semaphore is a counter with an associated queue that may contain waiting tasks. When a task examines the semaphore, it decrements the count by one and tests for a negative count. If the count is not negative, the task continues to execute. If the count is negative, some other task has set the semaphore; therefore the task is suspended and is placed on the queue for the semaphore.

When a task completes, or completes the portion of the task that is synchronized by the semaphore, it increments the count by one, activating the oldest task on the queue for the semaphore.

Typically, the initial value of the semaphore is one. When a task examines the semaphore, the semaphore is decremented and the value is zero. The task executes. If another task examines the semaphore, the semaphore value becomes negative and the second task is suspended. When the executing task completes, it increments the semaphore and the suspended task resumes execution. The value of the semaphore is now zero. Any task that examines the semaphore before the second task completes goes on the queue. If the task completes before another task examines the semaphore, the count is incremented again.

Semaphore operations also return the value of the semaphore, initialize the value, and modify the value.

The Semaphore Operations SVC (opcode >3D) performs the following operations:

- Signal — The calling task has completed the operation that must be synchronized.
- Wait — The calling task is ready to begin the operation that must be synchronized.
- Test — The calling task requests the value of the semaphore.
- Initialize — The calling task supplies a value for the semaphore.
- Modify — The calling task alters the value of the semaphore.

The supervisor call block for the SVC is as follows:

SVC >3D -- SEMAPHORE

ALIGN ON WORD BOUNDARY
CAN BE INITIATED AS AN EVENT

DEC	HEX		
0	0	>3D	<RETURN CODE>
2	2	SUB-OPCODE	SEMAPHORE NUMBER
4	4	SEMAPHORE VALUE	
6	6	RESERVED	

2279734

The call block contains the following:

Byte	Contents
0	Opcode, >3D.
1	Return code. DNOS returns zero when the operation completes satisfactorily. When the operation completes in error, DNOS returns an error code in this byte.
2	Sub-opcode: 00 — Signal. 01 — Wait. 02 — Test. 03 — Initialize. 04 — Modify.
3	Semaphore number. The number that identifies the Semaphore, in the range of 0 through 255.

Byte	Contents
4–5	Semaphore value. A value in the range of -128 through 127. Supplied by the user for Initialize and Modify operations. Returned by the system for Test operation.
6–7	[Reserved].

10.6.1.1 Signal. The Signal operation (sub-opcode >00) provides the synchronizing signal for a specified semaphore. Typically, a task executes a Signal operation when it has completed the portion of the task that requires synchronization of tasks. The Signal operation increments the semaphore value and activates the oldest task on the associated queue.

When the value of the semaphore is negative, its absolute value is the number of tasks queued for the semaphore. The Signal operation removes a task from the queue and updates the count correspondingly.

The following fields of the supervisor call block apply to the Signal operation:

- Opcode, >3D
- Return code
- Sub-opcode, >00
- Semaphore number

The following is an example of coding for a supervisor call block for a Signal operation:

```
        EVEN                    SIGNAL SEMAPHORE 5
SIG     BYTE >3D
        BYTE 0
        BYTE 0
        BYTE 5
        DATA 0,0
```

10.6.1.2 Wait. The Wait operation (sub-opcode >01) decrements the semaphore value and places the calling task on the associated queue when the value is negative.

When the value of the semaphore is positive, its value is the number of additional tasks that may execute concurrently. The Wait operation either places a task in execution or suspends the task awaiting its turn. Decrementing the count updates the count accordingly. When the result is positive or zero, the task continues to execute. Otherwise, the task is placed on the queue for the semaphore.

The following fields of the supervisor call block apply to the Wait operation:

- Opcode, >3D
- Return code

- Sub-opcode, >01
- Semaphore number

The following is an example of coding for a supervisor call block for a Wait operation:

```
           EVEN                         WAIT FOR SEMAPHORE 5
WAIT       BYTE >3D
           BYTE 0
           BYTE >01
           BYTE 5
           DATA 0,0
```

10.6.1.3 Test. The Test operation (sub-opcode >02) returns the value of the semaphore. A positive value is the number of tasks (or additional tasks) under control of the semaphore that may execute. A zero value indicates that the next task performing a wait operation is to be queued. A negative value shows how many tasks under control of the semaphore have been placed on the queue.

The following fields of the supervisor call block apply to the Test operation:

- Opcode, >3D
- Return code
- Sub-opcode, >02
- Semaphore number
- Semaphore value

The semaphore value in the range of -128 through 127 is returned in the semaphore value field.

The following is an example of coding for a supervisor call block for the Test operation:

```
           EVEN                         TEST SEMAPHORE 5
TEST       BYTE >3D
           BYTE 0
           BYTE >02
           BYTE 5
SVAL       DATA 0
           DATA 0
```

10.6.1.4 Initialize. The Initialize operation (sub-opcode >03) sets the semaphore to the specified value and activates suspended tasks on the semaphore queue according to the new value, as follows:

- When the new value is greater than or equal to zero, activate all tasks on the queue (if any).
- When the new value is negative but the old value is more negative, subtract the old value from the new value. The difference is the number of tasks to activate, beginning with the oldest task on the queue.

The following examples show the numbers of tasks activated in several cases:

		Number of Tasks	
Old Value	**New Value**	**On Queue**	**Activated**
Any	1	0	0
-1	2	1	1
-3	-1	3	2
-1	-2	1	0

The following fields of the supervisor call block apply to the Initialize operation:

- Opcode, >3D
- Return code
- Sub-opcode, >03
- Semaphore number
- Semaphore value

The semaphore value field contains the value to which the semaphore is initialized. The value is in the range of -128 through 127.

The following is an example of coding for a supervisor call block for an Initialize operation:

```
        EVEN                    INITIALIZE SEMAPHORE 5 TO
INIT    BYTE >3D                A VALUE OF 1
        BYTE 0
        BYTE >03
        BYTE 5
        DATA 1
        DATA 0
```

10.6.1.5 Modify. The Modify operation (sub-opcode >04) modifies the value of the semaphore by a specified amount and activates suspended tasks on the semaphore queue according to the new value, as follows:

- When the new value is greater than or equal to zero, activate all tasks on the queue.
- When the new value is negative but the old value is more negative, activate tasks, beginning with the oldest task on the queue. The number of tasks to activate is equal to the modifying value.

The following examples show the numbers of tasks activated in several cases:

			Number of Tasks	
Old Value	**Modifier**	**New Value**	**On Queue**	**Activated**
1	2	3	0	0
-1	2	1	1	1
-3	2	-1	3	2
-1	-2	-3	1	0

The Modify operation differs from the Initialize operation in that it alters the value of the semaphore by a fixed amount (positive or negative). An attempt to perform a Test operation, alter the value, and initialize the new value would provide unpredictable results because Signal or Wait operations by other tasks could intervene between the Test and Initialize operations.

The following fields of the supervisor call block apply to the Modify operation:

- Opcode, >3D
- Return code
- Sub-opcode, >04
- Semaphore number
- Semaphore value

The semaphore value field contains the value to be added to the semaphore value. The range of values for the semaphore value is -128 through 127; the modifier must not cause the semaphore value to violate these limits.

The following is an example of coding for a supervisor call block for a Modify operation:

```
        EVEN                   MODIFY SEMAPHORE 5 VALUE BY 2
MOD     BYTE >3D
        BYTE 0
        BYTE >04
        BYTE 5
        DATA 2
        DATA 0
```

10.6.2 Using Event Synchronization

Event synchronization allows a single task to synchronize I/O and semaphore operations within that task. Specifically, any operation of the I/O Operations SVC or the Semaphore Operations SVC may be initiated as an event. At any time prior to completion of the event, the task may be suspended pending completion of the event, or of any of a specified set of events.

An I/O or semaphore operation is initiated as an event when the Initiate Event SVC initiates the operation. It is not necessary to issue the I/O or semaphore SVC in the normal way; the Initiate Event SVC processor passes the call block for the operation to the appropriate SVC processor for execution.

The Wait for Event SVC specifies a set of events and suspends the calling task until one of these events completes. The SVC returns the flags for the specified set of events.

The Post Event SVC terminates a Wait for Event SVC that is issued by any task in any job except the system job. Thus, a single task can synchronize I/O and semaphore operations of tasks in different jobs.

10.6.2.1 Initiating an Event. The Initiate Event SVC (opcode >41) initiates any I/O or semaphore operation as an event. The supervisor call block specifies the address of the supervisor call block that defines the operation.

The supervisor call block for the SVC is as follows:

SVC >41 -- INITIATE EVENT   ALIGN ON WORD BOUNDARY

DEC	HEX		
0	0	>41	<RETURN CODE>
2	2	RESERVED	EVENT NUMBER
4	4	REQUEST BLOCK ADDRESS	
6	6	RESERVED	

2279735

The call block contains the following:

Byte	**Contents**
0	Opcode, >41.
1	Return code. DNOS returns zero when the operation completes satisfactorily. When the operation completes in error, DNOS returns an error code in this byte.
2	Reserved. Must be set to zero.
3	Event number. The number that identifies the event, in the range of 0 through 31.
4-5	Request block address. The address of the supervisor call block for the operation initiated as an event.
6-7	[Reserved].

DNOS initiates the requested operation and returns control to the calling task. If an operation other than an I/O operation or a semaphore operation is defined in the requested block, no operation is initiated, and DNOS returns an error code. If the requested operation is valid, DNOS initiates the operation and returns control to the calling task.

The following is an example of coding for a supervisor call block for an Initiate Event SVC, the Read operation it initiates, and the read buffer:

```
        EVEN               INITIATE I/O OPERATION IN CALL
IEVT    BYTE >41           BLOCK AT RDFI AS EVENT.
        BYTE 0             USE EVENT NUMBER 6.
        BYTE 0
EVTNO   BYTE 6
        DATA RDFI
        DATA 0
RDFI    DATA 0             READ A RECORD OF FILE ASSIGNED
        BYTE >09,>4C       TO LUNO >4C AND LOCK
        BYTE 0,>4C         THE RECORD
        DATA SFRB
        DATA 80
        DATA 0
SFRB    BSS 80             READ BUFFER
```

10.6.2.2 Waiting for Events. The Wait for Event SVC (opcode >42) specifies one or more events and suspends the calling task until one or more of the events has completed. Each event represents an operation or event that is initiated by an Initiate Event SVC, or an event that will be initiated by an Initiate Event SVC, or an event that is to be completed by a Post Event SVC. The set of events is specified by a mask (bit vector) consisting of two words. The mask provides for the full range of events, 0 through 31. DNOS returns a field of event bits indicating which events have completed.

The calling task specifies a maximum wait time. When none of the specified events completes within the maximum time, DNOS returns an error code to the task and the task resumes operation. If the maximum wait time is zero, the Wait for Event SVC acts as a polling operation that returns immediately and shows the events that have completed. If the maximum wait time is set to -1, no maximum time is specified for the calling task. In this case, the Wait for Event SVC does not time out before an event has completed.

The supervisor call block for the SVC is as follows:

SVC >42 — WAIT FOR EVENT ALIGN ON WORD BOUNDARY

DEC	HEX		
0	0	>42	<RETURN CODE>
2	2	MAXIMUM WAIT TIME	
4	4	<EVENT BITS>	
6	6		
8	8	EVENT MASK	
10	A		
14	E	RESERVED	

2279736

The call block contains the following:

Byte	Contents
0	Opcode, >42
1	Return code. DNOS returns zero when the operation completes satisfactorily. When the operation completes in error, DNOS returns an error code in this byte.
2-3	Maximum wait time, expressed as a number of 50-ms clock periods. If set to −1, no maximum wait time is specified.
4-7	Event bits. Returned by DNOS with a bit set for each selected event that has completed.

Byte	Contents
8-11	Event mask. Set bits to one to define the desired set of events for which to wait.
12-13	[Reserved].

The event mask and the event bits fields each consist of two words, one bit for each supported event number. Bits 0 through 15 of the first word of each field correspond to event numbers 0 through 15, respectively. Bits 0 through 15 of the second word correspond to event numbers 16 through 31, respectively. The following shows the event numbers that correspond to the bits of the two fields:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

2279737

The maximum wait time prevents delaying the task indefinitely when an equipment failure or error prevents normal termination of the event. The time selected should be greater than the maximum time the events would require to terminate normally.

The following is an example of coding for a supervisor call block for a Wait for Event SVC:

```
        EVEN               WAIT FOR EVENTS 1, 3, 4, AND 6
WFE     BYTE >42           TIMEOUT AFTER 500 MS
        BYTE 0
        DATA 10
EFLAG   DATA 0,0
        BYTE >5A00,0
        DATA 0
```

10.6.2.3 Posting an Event. The Post Event SVC (opcode >4F) sets a specified event bit to one to indicate that an event is complete and terminates any Wait for Event SVC that is outstanding for the specified event. The event bits can be set for any task of any job. The Post Event SVC indicates that an event is complete even if that event was not initiated.

The supervisor call block for the SVC is as follows:

SVC > 4F -- Post Event　　　　Align on word boundary

Dec	Hex		
0	0	>4F	<Return Code>
2	2	Run ID	Event Number
4	4	Job ID	
6	6	Reserved	

2283150

The call block contains the following:

Byte	Contents
0	Opcode, >4F.
1	Return code. DNOS returns zero when the operation completes satisfactorily. When the operation completes in error, DNOS returns an error code in this byte.
2	Run-time ID. The run-time ID of the task whose event bits are set.
3	Event number. This number identifies the event to be posted and is in the range of 0 through 31.
4–5	Job ID. The job ID of the job that contains the task whose event bits are set. Must not be the system job (job ID = 0).
6–7	Reserved. Must be set to zero.

The Post Event SVC provides intertask synchronization even if the cooperating tasks reside in different jobs. It is recommended that the cooperating tasks specify a particular event number and synchronize their operations by performing Wait for Event and Post Event SVCs to that event number. It is not necessary to initiate the events that are waited and posted.

The Post Event SVC or normal event completion leaves the event bit set to one. Specifying the Wait for Event SVC will immediately reset the event bit to zero.

The following is an example of the code for a supervisor call block for a Post Event SVC:

```
          EVEN                        POST EVENT 3 IN TASK >11, JOB >13
PEV       BYTE >4F                    OPCODE
          BYTE 0                      RETURN CODE
          BYTE >11                    TASK RUN-TIME ID
          BYTE 3                      EVENT NUMBER
          DATA >13                    JOB ID (CANNOT BE ZERO)
          DATA 0                      RESERVED
```

10.7 ACCESSING STATUS AND SYSTEM INFORMATION

DNOS provides SVCs that allow the task to access status and system information. Specifically, a task can:

- Access system date and time
- Obtain parameters
- Enter a message in the system log
- Obtain job and task identifiers
- Program end action
- Obtain task status
- Obtain return code data

10.7.1 Accessing System Date and Time

The system maintains the date and time to supply to any task. The date and time are entered when the first user logs on the system. A user may alter the date and time with an SCI command. A privileged task may alter the date and time by executing a Set Date and Time SVC. Any task may read the date and time by executing a Get Date and Time SVC.

The system stores the date and time as binary values; the year, date, hour, minute, and second occupy one word each. The date is a Julian (day of the year) date. The hour is the hour of a 24-hour day; the range of numbers is 0 through 23. To print these values the task may call Convert Binary to Decimal SVC for each binary value, move the ASCII characters to a buffer, and print the contents of the buffer. Further conversion of the date and hour is required to print the month and day and to print the hour as AM or PM.

10.7.1.1 Get Date and Time SVC. The system provides date and time support for all tasks. The Get Date and Time SVC (opcode >03) returns the binary date and time values in a five-word buffer at a specified address.

The supervisor call block for the SVC is as follows:

SVC >03 -- GET DATE AND TIME   ALIGN ON WORD BOUNDARY

DEC	HEX		
0	0	>03	<RETURN CODE>
2	2	BUFFER ADDRESS	

2279738

The call block contains the following:

Byte	Contents
0	Opcode, >03.
1	Return code. DNOS returns zero when the operation completes satisfactorily. When the operation completes in error, DNOS returns an error code in this byte.
2-3	Address of a five-word buffer into which DNOS places the date and time.

The following table shows the representation of a date and time in the five-word buffer. The time is 25 seconds past 3:24 PM on September 20, 1979:

	Year 0	Day 2	Hour 4	Minute 6	Second 8
Relative address	>07BB	>0107	>000F	>0018	>0019

The following is an example of coding for a supervisor call block for a Get Date and Time SVC, and for the buffer required for the operation:

```
DTBUF    BSS 10                OBTAIN SYSTEM DATE AND TIME IN
         EVEN                  BUFFER DTBUF.
GSDT     BYTE 3
         BYTE 0
         DATA DTBUF
```

10.7.1.2 Set Date and Time SVC. Initializing the system includes initializing the date and time. A software privileged task may issue a Set Date and Time SVC (opcode >3B) to set the date and time stored by the system. The values to which the date and time are set are binary values similar to those returned by the Get Time and Date SVC. These values are placed in a six-word buffer prior to issuing the SVC.

The supervisor call block for the SVC is as follows:

SVC >3B -- SET DATE AND TIME

ALIGN ON WORD BOUNDARY
PRIVILEGED TASK ONLY

DEC	HEX		
0	0	>3B	<RETURN CODE>
2	2	BUFFER ADDRESS	
4	4	RESERVED	

2279739

The call block contains the following:

Byte	Contents
0	Opcode, >3B.
1	Return code. DNOS returns zero when the operation completes satisfactorily. When the operation completes in error, DNOS returns an error code in this byte.
2-3	Address of a six-word buffer which contains the binary values to DNOS stores as the date and time.

The calling task places values of date and time in a six-word buffer as follows:

DEC	HEX	
0	0	YEAR
2	2	DAY OF THE YEAR
4	4	HOUR (0 THROUGH 23)
6	6	MINUTE (0 THROUGH 59)
8	8	SECOND (0 THROUGH 59)
10	A	RESERVED

2279740

The following is an example of coding for a supervisor call block for a Set Date and Time SVC, and for the buffer that contains the new values for the date and time:

```
DTBFF     DATA 1979              SET SYSTEM DATE AND TIME TO 8:47:15
          DATA 267               AM SEPT. 25, 1979. IN AN ACTUAL
          DATA 8                 APPLICATION PROGRAM THESE VALUES
          DATA 47                WOULD BE ENTERED AT A TERMINAL,
          DATA 15                CONVERTED TO BINARY, AND MOVED TO
          DATA 0                 THESE LOCATIONS.
          EVEN
IDATM     BYTE >3B
          BYTE 0
          DATA DTBFF
          DATA 0
```

10.7.2 Obtaining Parameters

The Bid Task and Scheduled Bid Task SVCs pass two parameters to the task being bid. The corresponding SCI commands also pass parameters. A task accesses the parameters supplied when it was bid by executing a Get Task Bid Parameters SVC (opcode >17). Each parameter consists of two bytes. The SVC returns the four bytes of parameters in the supervisor call block.

The supervisor call block for the SVC is as follows:

SVC >17 -- GET TASK BID PARAMETERS ALIGN ON WORD BOUNDARY

DEC	HEX		
0	0	>17	<RETURN CODE>
2	2	<PARAMETER 1>	
4	4	<PARAMETER 2>	

2279741

The call block contains the following:

Byte	Contents
0	Opcode, >17.
1	Return code.
2-3	DNOS returns parameter 1 (supplied when the calling task was bid) in this field.
4-5	DNOS returns parameter 2 (supplied when the calling task was bid) in this field.

The following is an example of coding for a supervisor call block for a Get Task Bid Parameters SVC:

```
         EVEN                       GET PARAMETERS ENTERED WHEN TASK
GTBP     BYTE >17                   WAS BID.
         BYTE 0
PAR1     DATA 0
PAR2     DATA 0
```

10.7.3 Logging a Message

A task may write a message in the system log by executing a System Log SVC (opcode >21). The message must be stored in a buffer that contains the length of the message in the first byte. The characters of the message occupy subsequent bytes of the buffer. The maximum message length is 255 characters.

The supervisor call block for the SVC is as follows:

SVC > 21 -- SYSTEM LOG   ALIGN ON WORD BOUNDARY

DEC	HEX		
0	0	>21	<RETURN CODE>
2	2	RESERVED	
4	4	BUFFER ADDRESS	
6	6	RESERVED	

2279742

The call block contains the following:

Byte	Contents
0	Opcode, >21.
1	Return code. DNOS returns zero when the operationzcompletes satisfactorily. When the operation completes in error, DNOS returns an error code in this byte.
2-3	Reserved.
4-5	Address of buffer that contains the length of the message in the first byte, followed by the characters of the message.
6-7	Reserved.

The following is an example of coding for a supervisor call block for a System Log SVC, and for the buffer that contains the message to be written to the system log:

```
MESS11    BYTE MSGEND-$-1
          TEXT 'TEXT OF MESSAGE'
MSGEND    EQU   $
          EVEN
SCBSL     BYTE >21                    WRITE MESSAGE IN BUFFER MESS11 TO
          BYTE 0                      SYSTEM LOG.
          DATA 0
          DATA MESS11
          DATA 0
```

10.7.4 Obtaining Task and Job Identifiers

A task can obtain its run-time and installed IDs, the ID of the station under which it is executing, and the ID of the job under which it is executing.

To obtain these IDs, the task executes the Self-Identification SVC (opcode >2E). The installed ID is the ID that identifies the task on the program file. The run-time ID is the identifier assigned to the task when it began execution. The station ID is specified in the SCI command or SVC that initiated execution of the task. Interaction with the user is by means of that station (terminal). The job ID is the identifier of the current job.

The supervisor call block for the SVC is as follows:

SVC > 2E -- SELF-IDENTIFICATION

DEC	HEX		
0	0	> 2E	<RUN-TIME ID>
2	2	<INSTALLED ID>	<STATION ID>
4	4	<JOB ID>	

2279743

The call block contains the following:

Byte	Contents
0	Opcode, >2E.
1	Run-time ID. Returned by the system.
2	Installed ID. Returned by the system.
3	Station ID. Returned by the system.
4-5	Job ID. Returned by the system.

The following is an example of coding for a supervisor call block for a Self-Identification SVC:

```
SID       BYTE >2E              OBTAIN IDENTIFIERS OF CALLING TASK
RTID      BYTE 0
INSTID    BYTE 0
STID      BYTE 0
JID       DATA 0
```

10.7.5 Programming End Action

A task executes the end action routine at the address in the third word of the task when execution terminates abnormally. The end action routine should identify the error that caused abnormal termination, and take corrective action when appropriate, or terminate. DNOS provides two supervisor calls that may be used in end action routines.

10.7.5.1 Get End Action Status SVC. The end action routine of a task often requires status information and the error code of the error that caused termination. The Get End Action Status SVC (opcode >2F) returns the workspace pointer, the program counter contents, and the status register contents that indicate the environment of the terminating error. The call also returns the error code of the terminating error.

The supervisor call block for the SVC is as follows:

SVC > 2F -- GET END ACTION STATUS   ALIGN ON WORD BOUNDARY

DEC	HEX		
0	0	>2F	<ERROR CODE>
2	2	<WORKSPACE POINTER>	
4	4	<PROGRAM COUNTER>	
6	6	<STATUS REGISTER>	
8	8	RESERVED	

2279744

The call block contains the following:

Byte	Contents
0	Opcode, >2F.
1	Error code. DNOS returns the code of the error that caused abnormal termination.
2-3	DNOS returns the address of the workspace in effect when the error occurred.
4-5	DNOS returns the contents of the program counter when the error occurred.

Byte	Contents
6–7	DNOS returns the contents of the status register when the error occurred.
8–9	Reserved.

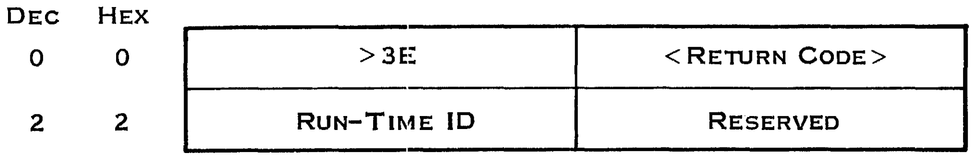
The following is an example of coding for a supervisor call block for a Get End Action Status SVC:

```
        EVEN                GET STATUS OF TASK WHEN END ACTION IS
GEAS    BYTE >2F            BEING TAKEN.
TERR    BYTE 0
WPREG   DATA 0
PCREG   DATA 0
STREG   DATA 0
        DATA 0
```

10.7.5.2 Reset End Action Status SVC. Normally, a task may take end action only once. However, an end action routine may be able to correct an error and allow the task to continue. The Reset End Action Status SVC (opcode >3E) resets a flag in the task status block to allow end action to be taken again. This SVC is appropriate in an end action routine that allows the task to continue. The SVC may be executed to reset end action for another task (other than the calling task). When the SVC is issued, and end action has either not been taken or has been reset, the SVC is effectively ignored. Also, the SVC is effectively ignored when the task was terminated by execution of a Kill Task SVC or a Kill Task SCI command.

The supervisor call block for the SVC is as follows:

SVC >3E -- RESET END ACTION STATUS



DEC	HEX		
0	0	>3E	<RETURN CODE>
2	2	RUN-TIME ID	RESERVED

2279745

The call block contains the following:

Byte	Contents
0	Opcode, >3E.
1	Return code. DNOS returns zero when the operation completes satisfactorily. When the operation completes in error, DNOS returns an error code in this byte.
2	Run-time ID of task for which end action is to be reset, or zero to reset end action for the calling task.
3	Reserved.

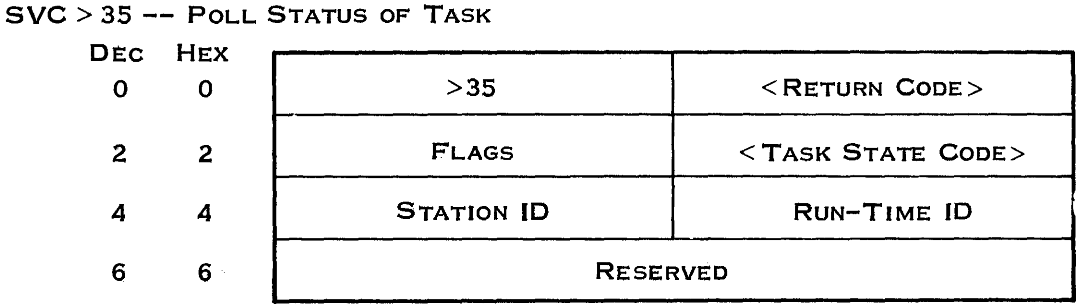
The following is an example of coding for a supervisor call block for a Reset End Action Status SVC:

```
REAS     BYTE >3E               RESET END ACTION FOR CALLING TASK.
RCODE    BYTE 0
RTID     DATA 0
```

10.7.6 Obtaining Task Status

A task can obtain the task status code for itself or for any task in the same job. Task status codes are listed in Table 10-1. The Poll Status of Task SVC (opcode >35) returns the task status code of the task in a byte of the supervisor call block. The SVC searches a list of tasks to locate the run-time ID specified. When the run-time ID is found, the SVC compares the station ID of the task to the station ID specified in the call block. The SVC may specify any task in the job, any task assigned to a terminal (station), or any task assigned to the same terminal as the calling task.

The supervisor call block for the SVC is as follows:



SVC > 35 -- POLL STATUS OF TASK

DEC	HEX		
0	0	>35	<RETURN CODE>
2	2	FLAGS	<TASK STATE CODE>
4	4	STATION ID	RUN-TIME ID
6	6	RESERVED	

2279746

The call block contains the following:

Byte	**Contents**
0	Opcode, >35.
1	Return code. DNOS returns zero when the operation completes satisfactorily. When the operation completes in error, DNOS returns an error code in this byte.
2	Flags: Bit 0 — Set to 1 when station ID is specified in byte 4. Set to 0 when station ID is the station ID of the calling task. Bits 1–7 — Reserved.
3	DNOS returns the task state code in this byte.
4	Station ID (ignored when user flag bit 0 is set to 0). DNOS polls tasks assigned to station ID, or all tasks in the job when this field contains >FF.
5	Run-time ID of task for which the task state code is required.
6–7	Reserved.

The following is an example of coding for a supervisor call block for a Poll Status of Task SVC:

```
PSTC     BYTE >35           GET TASK STATE CODE OF TASK WITH
RCDE     BYTE 0             RUN-TIME ID OF >3F ASSIGNED TO
FLGTS    BYTE >80           TERMINAL WITH STATION ID 8.
TSKSC    BYTE 0
STAN     BYTE 8
TRID     BYTE >3F
         DATA 0
```

Table 10-1. Task State Codes

Code (Hexadecimal)	Significance
00	In ready state
01	Awaiting memory
02	Job in a nonexecutable state
03	Task being loaded into memory
04	Terminated
05	In time delay
06	Unconditionally suspended
07	Awaiting Ten-X processor completion
09	Suspended for I/O
0F	Suspended for aborted I/O
14	Awaiting overlay load services
17	Awaiting co-routine activation
19	Awaiting completion of initiated I/O
1E	Awaiting system semaphore
1F	Awaiting scheduled task bid
20	Awaiting install volume completion
22	Awaiting disk management services
24	Awaiting queue input
25	Awaiting task installation
26	Awaiting procedure installation
27	Awaiting overlay installation
28	Awaiting completion of task deletion
29	Awaiting completion of procedure deletion
2A	Awaiting completion of overlay deletion
2B	Suspended by Bid Task SVC
2D	Awaiting read/write task completion
30	Awaiting system table area
31	Awaiting map program name to ID completion
34	Awaiting unload volume completion
36	Awaiting any I/O
37	Awaiting assignment of program file space
38	Awaiting initialize new volume completion
3D	Suspended by Semaphore SVC
40	Awaiting segment management services
42	Awaiting event completion
43	Awaiting name management services
48	Awaiting job management services
4A	Awaiting forced swap completion
4C	Awaiting return code processing completion

10.7.7 Obtaining Return Code Data

When an SVC is executed, it may return an error or status code in byte 1. The nature of the error or status may require that some data from the call block of the SVC that returns the code be available to any task that builds a meaningful error message. The Return Code Processing SVC (opcode >4C) returns appropriate data in a specified buffer. The SVC supplies the address of the call block that contains the error or status code. DNOS accesses the code in that block and returns the data from the block as selected by an internal table of codes. The system also returns the number of the message corresponding to the code.

The supervisor call block for the SVC is as follows:

SVC >4C -- RETURN CODE PROCESSING ALIGN ON WORD BOUNDARY

DEC	HEX		
0	0	>4C	<RETURN CODE>
2	2	SVC BLOCK ADDRESS	
4	4	BUFFER ADDRESS	
6	6	<MESSAGE NUMBER>	
8	8	RESERVED	

2279747

The call block contains the following:

Byte	Contents
0	Opcode, >4C.
1	Return code. DNOS returns zero when the operation completes satisfactorily. When the operation completes in error, DNOS returns an error code in this byte.
2-3	SVC block address. The address of the call block that contains the return code to be processed.
4-5	Buffer address. Address of a buffer into which the SVC places data from the call block.
6-7	<Message number>. DNOS returns the number of the error or warning message that corresponds to the return code.
8-9	Reserved.

The return code in byte 1 is that of the Return Code Processing SVC, not the code to be processed.

The SVC block address in bytes 2-3 is the address of the call block that contains the code to be processed. The contents of this block must not be altered after the unsuccessful completion of the SVC until the Return Code Processing SVC has completed.

The buffer address, bytes 4-5, is the address of the buffer in which the SVC returns the data. The first byte of the buffer must contain the buffer size, in bytes. The buffer should contain a minimum of 50 bytes.

The message number returned by DNOS in bytes 6-7 is the number by which the message corresponding to the return code is identified. This number can be used to access the message in the DNOS error message file or in the *DNOS Messages and Codes Manual.* A task can directly use the DNOS error message files by an application program that uses the interface routines S$TERM and S$CMSG. These routines are described in the *DNOS Systems Programmer's Guide.*

At the completion of the Return Code Processing SVC, the first byte of the buffer contains the number of characters returned. The data from the relevant fields of the process call block follows, with semicolons separating the fields of the call block.

The following is an example of coding for a supervisor call block for a Return Code Processing SVC and the required buffer:

```
RCPSVC   DATA >4C00          PROCESS RETURN CODE IN CALL.
         DATA IOSVC          BLOCK IOSVC.
         DATA BUFFER         BUFFER ADDRESS.
MESSNO   DATA 0
         DATA 0
BUFFER   BYTE BUFEND-$
BUFTXT   BSS 50
BUFEND   EQU  $
```

11

System Interface

11.1 SPECIAL SYSTEM SERVICES

This section describes a group of supervisor calls that interface with the system for special services. The SVCs in the group are:

- Retrieve System Data
- Disk Management
- Suspend for Queue Input
- Read/Write TSB
- Read/Write Task

Only system tasks are allowed to execute the Suspend for Queue Input SVC, and only software privileged tasks may execute Disk Management, Read/Write TSB, and Read/Write Task SVCs.

11.2 OBTAINING SYSTEM DATA

Many of the system data structures shown in the *DNOS System Design Document* may be accessed by a task using the Retrieve System Data SVC (opcode >3F). The SVC returns a specified portion of any of the following system data structures:

- PDTs
- TILINE* peripheral control space (TPCS)
- CSEG area LGLCOM
- CSEG area NFCLKD
- CSEG area NFDATA
- CSEG area NFPTR
- CSEG area PMDATA
- CSEG area NFJOBC

*TILINE is a registered trademark of Texas Instruments Incorporated.

The contents of the supervisor call block define the structure to be accessed, the portion to be returned, and the number of words to be returned. The user must be familiar with the desired structure in order to use the SVC. The SVC returns the requested data to the calling task; it does not allow the task to modify the contents of system data structures.

SVC >3F -- Retrieve System Data   Align on word boundary

Dec	Hex		
0	0	>3F	<Return Code>
2	2	Data Structure Type	Flags
4	4	Index	
6	6	Offset (Bytes)	
8	8	Buffer Length (Bytes)	
10	A	Length Returned (Bytes)	
12	C	Buffer Address	
14	E	Reserved	

2279749

The call block contains the following:

Byte **Contents**

0 Opcode, >3F.

1 Return code. DNOS returns zero when the operation completes satisfactorily. When the operation completes in error, DNOS returns an error code in this byte.

2 Data structure type. Enter the appropriate code, as follows:

>81	PDT (Physical Device Table)
>82	TPCS (TILINE Peripheral Device Space)
>83	CSEG LGLCOM (Log Common Data)
>84	CSEG NFCLKD (Clock Data)
>85	CSEG NFDATA (Nucleus Data)
>86	CSEG NFPTR (System Pointers)
>87	CSEG PMDATA (Program Management Data)
>88	CSEG NFJOBC (Job Management Data)

Byte	Contents
3	Flags: Bit 0 — Block format flag. Set as follows: 1 — Short format block. Returns one word, in bytes 10 and 11. 0 — Long format block. Returns data in a specified buffer. Bit 1 — Address flag. Set as follows: 1 — Value in bytes 6-7 is to be used as an indirect address. 0 — Value in bytes 6-7 is to be used as an offset (relative to first byte of structure).
4-5	Index. The number of a specific data structure of the type specified in byte 2. Applies to PDTs.
6-7	Address of requested data within the specified data structure. When address flag is set to 1, address is an indirect address. Otherwise, address is an offset (relative address) from the first byte of the data structure.
8-9	Buffer length. The length of the buffer in which system returns requested data. Set to zero when block format flag is set to one.
10-11	When block format flag is zero, the system returns the number of bytes of data stored in the buffer. When block format flag is set to one, system returns the data from the specified address of data structure. When the block format and return address flags are set to one, DNOS returns the address of the data structure here.
12-13	Buffer address. Address of buffer for data requested from the system. Set to zero when block format flag is set.
14-15	Reserved.

The data structure type selected controls the content of certain fields of the call block. The index field, bytes 4-5, applies to data structure type >81, physical device table. The system includes one PDT for each device. The contents of the index field specify which PDT to access. The field is not used for other data structures.

Most of these data structures consist of data values and pointers. When accessing a data value or pointer in the PDT or TPCS, use an offset (address flag set to 0). Data structure types >83 through >88 specify CSEG structures of DNOS. When accessing a data value via a pointer in the CSEG, use an indirect address (address flag set to 1).

Since some PDTs are alternates for a master PDT, the maximum size of a PDT varies. Attempting to access a field not defined in an alternate PDT returns an error code.

The following is an example of coding for a supervisor call block for a Retrieve System Data SVC and for the buffer in which the data is returned:

```
         EVEN                    GET FIRST TEN BYTES OF PDT 1
RSD      BYTE >3F                IN BUFFER AT PDTDAT
         BYTE 0
         BYTE >81
         BYTE 0
         DATA 1
         DATA 0
         DATA 10
BLEN     DATA 0
         DATA PDTDAT
         DATA 0
PDTDAT   BSS 10
```

11.3 ALLOCATING AND DEALLOCATING DISK SPACE

When performing file I/O as described in Section 5, you need not allocate or deallocate disk space for the file; file management automatically allocates and deallocates the space. Space for writing directly to a disk must be allocated, and should be deallocated when it is no longer required. To allocate or deallocate space on a disk or on a double-sided, double-density diskette, request the services of the disk manager by executing a Disk Management SVC (opcode >22). The disk manager allocates and deallocates space in allocatable disk units (ADUs). An ADU, by definition, is the smallest amount of disk storage that can be allocated on a disk. An ADU consists of an integer number of sectors; the number varies according to the type of disk. The numbers of sectors per ADU for the types of disks supported by DNOS are shown in Table 11-1. These numbers were chosen to provide no more than 65,535 ADUs per disk; the ADU address is 16 bits or less.

Four sub-opcodes specify the type of operation to be performed. Sub-opcode 0 requests deallocation of a specified number of ADUs starting at a specified ADU. Sub-opcode 1 requests allocation of a specified number of blocks of a specified number of bytes each. When that large an area is not available, disk manager returns an error code. Sub-opcode 2 also requests allocation of a specified number of blocks of a specified size. When the requested space is not available, disk manager allocates the space available and does not return an error code. Disk manager allocates space for sub-opcodes 1 and 2 as close as possible to the specified ADU location. When ADU 0 is requested, the lowest available area is allocated. Sub-opcode 3 requests allocation of a specified number of blocks of a specified size at the specified ADU location. When the specified ADU is not available, disk manager returns an error code. When the total area requested is not available at the specified ADU location, disk manager allocates the space available, and does not return an error code. Disk manager allocates space as a contiguous area of disk.

Access to a disk for allocation or deallocation requires the address of the PDT for the disk. This may be obtained by executing a Retrieve System Data SVC.

Table 11-1. Disk Descriptions

Disk Type	Number of Sectors	Number of ADUs	Sectors per ADU	Bytes per ADU
FD1000	4,004	4,004	1	288
DS31	9,744	9,744	1	288
DS32	9,744	9,744	1	288
DS10	16,320	16,320	1	288
DS25	77,520	25,840	3	864
DS50	154,850	51,616	3	864
DS80	244,915	40,819	6	1,536
DS200	588,430	65,381	9	2,592
DS300	930,677	62,045	15	3,840
WD800/18	72,261	24,087	3	768
WD800/43	168,609	56,203	3	768

SVC >22 -- Disk Management

Align on word boundary
Privileged task only

Dec	Hex		
0	0	>22	<Return Code>
2	2	Sub-Opcode	Reserved
4	4	Disk PDT Address	
6	6	Block Size (Bytes)	
8	8	Number of Blocks or Number of ADUS	
10	A		
12	C	ADU Number	

2279750

The call block contains the following:

Byte	Contents
0	Opcode, >22.
1	Return code. DNOS returns zero when the operation completes satisfactorily. When the operation completes in error, DNOS returns an error code.
2	Sub-opcode, as follows: 0 — Deallocate disk space. 1 — Allocate requested number of blocks of specified size, or return an error code. 2 — Allocate requested number of blocks of specified size, or as much as is available. 3 — Allocate requested number of blocks of specified size (or as much as is available), at the requested ADU. Return an error code when the ADU is not available.
3	Reserved.
4–5	Address of the physical device table (PDT) for the disk on which space is to be allocated or deallocated.
6–7	For allocation, size (in bytes) of blocks. Ignored for deallocation.
8–11	For allocation, number of blocks to be allocated. For deallocation, bytes 8–9 are ignored and bytes 10–11 contain the number of ADUs to deallocate.
12–13	For allocation, the requested ADU number. For deallocation, the starting ADU of the area to be deallocated.

At the completion of a successful allocation, bytes 8 through 13 of the supervisor call block contain the following allocation values:

Dec	Hex		
8	8	ADU/Block	Blocks/ADU
10	A	Number of ADUS Allocated	
12	C	Starting ADU Number	

2279751

The fields returned define the allocated area, as follows:

Byte	Contents
8	Number of ADUs allocated per block (set to 1 if less than 1).
9	Number of blocks per allocated ADU (set to 0 if less than 1).
10–11	Number of ADUs allocated.
12–13	ADU number of the first ADU in the allocated area.

The following is an example of coding for a supervisor call block for a Disk Management SVC, assuming the address of the PDT has been moved to the call block after having been obtained with a Retrieve System Data SVC:

```
        EVEN          ALLOCATE AS MUCH DISK SPACE AS
DSCMGR  BYTE >22      POSSIBLE UP TO 64 128-BYTE BLOCKS
ERRCOD  BYTE 0        ON THE DISK STARTING AT LOWEST-
        BYTE 2        NUMBERED AVAILABLE ADU. THE PDT
        BYTE 0        ADDRESS IS PLACED IN LOCATION
DPDT    DATA 0        DPDT BY OTHER CODE.
        DATA 128
ADUBLK  BYTE 0
BLKADU  BYTE 0
SIZE    DATA 64
START   DATA 0
```

11.4 SUSPENDING A QUEUE-DRIVEN TASK

A task that is driven by a queue need not terminate, but may be suspended pending another entry in its queue. That is, when a queue-driven task has processed all current entries in the queue, it may either terminate or be suspended awaiting another entry in the queue. When the task terminates, it must be bid again before it can process a queue entry. When it suspends, it is automatically reactivated when an entry is placed in the queue. The Suspend for Queue Input SVC (opcode >24) suspends a queue-driven task until some other task places an entry in the queue. This SVC serves the same purpose, for a queue-driven task, as the End of Task SVC, except that the suspended task is not terminated. The suspended task is not necessarily rolled out, but may be chosen for rollout by task management.

CAUTION

This SVC can interfere with normal system operation unless the calling task performs additional processing. The calling task must execute an Extend Time Slice SVC (or otherwise disable task scheduling), check the queue for any entry that might have been made during processing of last entry, and then execute this SVC. The scheduler must remain disabled while this is done.

The supervisor call block for the SVC is as follows:

SVC >24 -- SUSPEND FOR QUEUE INPUT   SYSTEM TASK ONLY

DEC	HEX	
0	0	>24

2279752

The call block contains the following:

Byte	**Contents**
0	Opcode, >24

The following is an example of coding for a supervisor call block for a Suspend for Queue Input SVC:

```
SAQI     BYTE >24                   SUSPEND CALLING TASK AWAITING QUEUE
                                    INPUT.
```

11.5 ACCESSING THE TASK STATUS BLOCK

The task status block (TSB) is a DNOS data structure. Each task in the system is represented by a TSB. The Read/Write TSB SVC (opcode >2C) reads or writes a specified word in the TSB of a specified task. Access to TSBs is limited to those of tasks in the same job as the calling task. Do not attempt to execute this SVC without first examining the structure of the TSB (described in the *DNOS System Design Document*).

CAUTION

The Read/Write TSB SVC processor does not prevent alteration of the TSB. Such alteration of the TSB could interfere with continued normal system operation.

The normal sequence of SVCs for altering the TSB is as follows:

1. Execute an Extend Time Slice SVC for the task that is altering the TSB.
2. Read a value from the TSB.
3. Write a new value to the TSB.

The time slice should be extended while accessing the TSB of an active task because the contents of the TSB of an active task are continually being changed. The TSB contents read in step 2 might not be valid at the time the write function in step 3 is performed if the altering task is suspended between steps 2 and 3. The extend period should allow time for steps 2 and 3, and for the processing of alterations performed between these steps.

The supervisor call block for the SVC is as follows:

SVC > 2C -- Read/Write TSB

Align on word boundary
Privileged task only

Def	Hex		
0	0	>2C	<Return Code>
2	2	Flags	
4	4	Task Run ID	Offset into TSB
6	6	Value	
8	8	Reserved	

2279753

The call block contains the following:

Byte	Contents
0	Opcode, >2C
1	Return code. DNOS returns zero when the operation completes satisfactorily. When the operation completes in error, DNOS returns an error code.
2-3	Flags: Bits 0–14 — Reserved. Bit 15 — Set to 1 for write, to 0 for read.
4	Run-time ID of task whose TSB is to be accessed. Must be in the same job as the calling task.
5	TSB offset. Relative address within the TSB of the word to be accessed. It must be a byte address and should be an even number.
6-7	Value. For a read, DNOS places the value read from the TSB in this field. For a write, contents of this field are written to the TSB.
8-9	Reserved.

The following is an example of coding for a supervisor call block for a Read/Write TSB SVC:

```
         EVEN                    READ WORD AT BYTE OFFSET 10 IN TSB OF
RWTTSB   BYTE >2C                THE TASK WITH RUN-TIME ID OF >83.
         BYTE 0
SVCFLG   DATA 0
         BYTE >83
         BYTE 10
VALUE    DATA 0
         DATA 0
```

The following additional code illustrates the context in which the Read/Write TSB SVC is used:

```
         DXOP SVC,15             THIS EXAMPLE SHOWS AN EXTEND TIME
         EVEN                    SLICE SVC, A READ TSB SVC, AND A
EXTEND   BYTE 9,4                WRITE TSB SVC. CODE TO PROCESS THE
WRTFLG   DATA 1                  VALUE READ FROM THE TSB WOULD
          .                      FOLLOW THE READ TSB CALL.
          .
          .
         SVC @EXTEND
         SVC @RWTTSB
          .
          .
          .
         MOV @WRTFLG,@SVCFLG
         SVC @RWTTSB
```

11.6 ACCESSING TASK DATA

Under DNOS, a task can access data within another task of the same job. The task may be an application task or the system root. The system root contains:

1. Interrupt and XOP vectors
2. Nucleus common support routines
3. Common data segments
4. System table area

The system root is described in detail in the *DNOS System Design Document*.

CAUTION

If this SVC is used to modify a system overlay, system operation may cease.

The Read/Write Task SVC (opcode >2D) reads or writes as many as 16 words of data from or to a task in memory.

The supervisor call block for the SVC is as follows:

SVC > 2D -- Read/Write Task

Align on word boundary
Privileged task only

Dec	Hex		
0	0	>2D	<Return Code>
2	2	Task Run ID	Flags
4	4	Address	
6	6	Buffer (16 words)	
36	24		
38	26	Reserved	

2279754

The call block contains the following:

Byte	Contents
0	Opcode, >2D.
1	Return code. DNOS returns zero when the operation completes satisfactorily. When the operation completes in error, DNOS returns an error code.
2	Run-time ID of task to be accessed. Must be in the same job as the calling task. Set to 0 to access the system root.
3	Flags: Bit 0 — Set to 1 for write, to 0 for read. Bit 1 — ID flag. 0 — Task run ID specified in byte 2 1 — System overlay ID specified in byte 2 Bits 2-3 — Reserved Bits 4-7 — Number of words to be read or written; specify 0 to read or write 16 words
4-5	Relative address in the specified task of the first word of data to be read or written.
6-37	16-word buffer. DNOS places the data read in this buffer. For a write, contents of this buffer are written.
38-39	Reserved.

The following is an example of coding for a supervisor call block for a Read/Write Task SVC:

```
         EVEN               READ WORKSPACE (16 WORDS) OF THE TASK
RDTSK    BYTE >2D           WITH RUN-TIME ID OF >83. ASSUME
         BYTE 0             WORKSPACE IS AT ADDRESS 6.
         BYTE >83
         BYTE 0
         DATA 6
DATA     BSS 32
         DATA 0
```

12

SVC Compatibility

12.1 OTHER OPERATING SYSTEMS

This section describes four supervisor calls that are included in DNOS for compatibility with other operating systems. You may execute a program that was written to be executed under an earlier operating system without altering the program to replace these supervisor calls with others more appropriate for DNOS. You should use other supervisor calls that exploit the capabilities of DNOS in programs written to be executed under DNOS. The supervisor calls are:

- Get Common Data Address
- Return Common Data Address
- Put Data
- Get Data

These SVCs may be omitted when generating a system. Groups of optional SVCs are defined for system generation. The COMMON group consists of the Get Common Data Address and the Return Common Data Address SVCs. The INTERTASK group consists of the Put Data and Get Data SVCs. When a task executing under a system that does not include a group of SVCs calls an SVC in that group, DNOS returns >F0 in the return code byte of the supervisor call block.

The intertask common memory area must be defined when a system is generated. When the COMMON group of SVCs is to be included, also supply the required size of the intertask common memory area during system generation.

12.2 GET COMMON DATA ADDRESS SVC

The Get Common Data Address SVC (opcode >10) obtains the address of the beginning of the intertask common memory area and returns that address in workspace register 9. The SVC also returns the size of the area in workspace register 8.

The supervisor call block for the SVC is as follows:

SVC > 10 -- GET COMMON DATA ADDRESS

DEC	HEX		
0	0	>10	<RETURN CODE>

2279755

The call block contains the following:

Byte	Contents
0	Opcode, >10.
1	Return code. DNOS returns zero when the operation completes satisfactorily. When the operation completes in error, DNOS returns an error code in this byte.

DNOS offers several alternatives to the use of an intertask common area. Some alternatives are:

- Transferring common data through interprocess communication.
- Using a shared segment to pass data.

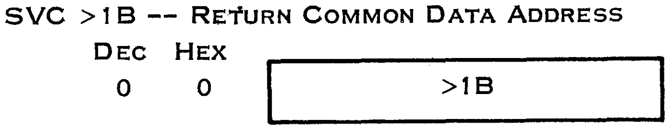
The following is an example of coding for a supervisor call block for a Get Common Data Address SVC:

```
SCBD     BYTE >10,0                GET THE ADDRESS OF INTERTASK COMMON.
```

12.3 RETURN COMMON DATA ADDRESS SVC

The Return Common Data Address SVC (opcode >1B) releases the intertask common memory obtained by the Get Common Data Address SVC. One more segment of memory is available to the program when intertask common memory is released.

The supervisor call block for the SVC is as follows:



2279756

The call block contains the following:

Byte	Contents
0	Opcode, >1B.

The following is an example of coding for a supervisor call block for a Return Common Data Address SVC:

```
SCBE      BYTE >1B                RELEASE THE MEMORY SEGMENT USED FOR
                                  INTERTASK COMMON.
```

12.4 PUT DATA SVC

The Put Data SVC (opcode >1C) transfers messages from a buffer in the calling task to the DNOS dynamic memory area for subsequent retrieval by a Get Data SVC. The identifier of the message is also the identifier of the queue in memory into which the message is placed, in a first-in, first-out manner.

The supervisor call block for the SVC is as follows:

SVC >1C -- PUT DATA   ALIGN ON WORD BOUNDARY

DEC	HEX		
0	0	>1C	<RETURN CODE>
2	2	RESERVED	MESSAGE ID
4	4	BUFFER ADDRESS	
6	6	[RESERVED]	
8	8	MESSAGE LENGTH	
10	A	RESERVED	

2279757

The call block contains the following:

Byte	Contents
0	Opcode, >1C.
1	Return code. DNOS returns zero when the operation completes satisfactorily. When the operation completes in error, DNOS returns an error code in this byte.
2	Reserved.
3	Message identifier.
4–5	Address of the buffer in the task that contains the message. The address must be an even (word boundary) address.
6–7	[Reserved].
8–9	Length of the buffer that contains the message. The length must be an even number of characters.
10–11	Reserved.

The preferred alternative to the use of Put Data to send a message to another task is to use the interprocess communication capability of DNOS.

The following is an example of coding for a message buffer, and a supervisor call block for a Put Data SVC:

```
BUFF2   TEXT 'DATA TO BE SENT TO ANOTHER TASK IS PLACED'
        TEXT 'IN A BUFFER IN THE CALLING TASK.
         .
         .
         .
        EVEN                     PLACE THE MESSAGE IN THE BUFFER AT
SCBG    BYTE >1C                 BUFF2 IN THE MESSAGE QUEUE FOR
        BYTE 0                   IDENTIFIER >1F.
        BYTE 0
        BYTE >1F
        DATA BUFF2
        DATA 0
        DATA 80
        DATA 0
```

12.5 GET DATA SVC

The Get Data SVC (opcode >1D) retrieves a message placed in the DNOS dynamic memory area by a Put Data SVC and moves the message to the specified buffer in the calling task. The message returned is the oldest message on a message queue corresponding to the message identifier specified in the call. Alternatively, the SVC deletes all messages in the queue without returning any.

The supervisor call block for the SVC is as follows:

SVC >1D -- GET DATA   ALIGN ON WORD BOUNDARY

DEC	HEX		
0	0	>1D	<RETURN CODE>
2	2	FLAGS	MESSAGE ID
4	4	BUFFER ADDRESS	
6	6	BUFFER LENGTH	
8	8	ACTUAL MESSAGE LENGTH	
10	A	RESERVED	

2279758

SVC **Paragraph**

SVC **Paragraph**

SVC **Paragraph**

SVC **Paragraph**

SVC **Paragraph**

SVC **Paragraph**

Appendix B

ASCII Device I/O Operations Tables

B.1 GENERAL

DNOS supports ASCII I/O operations with all devices. This appendix consists of the following tables that show key codes generated by DNOS supported terminals and an explanation of what the DSRs do with the codes according to the requested operation, tables that show graphics character sets, and a table that shows KIF collating sequences according to country code.

B.2 CHARACTER TYPES

Table B-1 through Table B-6 show how the DSR treats the ASCII characters for each key on the 911 VDT, 931 VDT, 940 VDT, Business System terminal, 820 KSR, and 783 TPD, respectively. The 783 TPD is representative of the TPD class of terminals. The column headings for each table mean the following:

ASCII Character

The defined ASCII character. Characters >00 through >1F and >7F are control characters and are not printable.

Terminal Key

The key you press on the terminal to generate the ASCII character.

Terminal Generated Code

The hexadecimal code generated by the terminal for that key. In the notation A/B, A is the code when the terminal is set to transmit even parity, and B is the code when the terminal is set to transmit odd parity. For the TPD DSR, this code is returned to your task when the device is in eight-bit ASCII mode and your task issues a Read Direct operation with the eight-bit data option. In pass thru mode on the 931, 940, and Business System terminals, this code is returned to your task, except for DC1 and DC3, which the DSR never returns. The 931, 940, and Business System terminals generate two or three character escape sequences for some keys. For example, the F1 key on the 931 generates 1B6931 (ESC i 1).

Type

Indicates one of the following:

S = System edit character
E = Event key
T = Task edit character
H = Hold character
F = End-of-file
R = End-of-record

ASCII Task Edit

Lists the codes returned to your task for this key when your task is doing a Read ASCII operation with the task edit flag set and the device has been opened in event key mode.

Buffer

The code placed in the read buffer. If there is no entry, no code is placed in the read buffer.

Event

The code placed in the event byte of the call block. If there is no entry, no code is placed in the event byte.

Flag

The flag or flags set in the system flags byte of the call block when this key is received. EVT means the event flag is set. EOF means the end-of-file flag is set.

ASCII

The code returned to your task for this key when your task is doing a simple Read ASCII operation with no special flags set. If there is no entry in this column, nothing is returned.

Direct

The code returned to your task for this key when your task is doing a Read Direct operation with no special flags set. This column is not applicable to a 911, 931, 940, or Business System terminal since a Read Direct on these terminals does not interact with the keyboard.

Figure B-1 through Figure B-3 show graphics character sets. To read or write graphics characters, an extended call block must be used with the graphics bit set in the extended flags. Figure B-1 shows the keyboard positions of the keys that produce graphics on the 911.

Figure B-2 shows the graphics characters for the 931. The chart on the left shows the characters and the ASCII codes they produce. For example, an uppercase A produces ASCII code >41. The top row contains the first digit of the code; the left column contains the second digit. If you are using the special character set, the chart on the right shows the graphics characters and the ASCII codes they produce. For example, the question mark (?) key produces the graphics character associated with the ASCII code >3F.

Figure B-3 shows the graphics characters for the 940. The chart on the left shows the graphics characters and the ASCII codes they produce. For example, a left arrow (←) produces ASCII code >4E. The top row contains the first digit of the code; the left column contains the second digit. The chart on the left shows the keyboard positions of the keys that produce these graphics characters.

Table B-1. 911 VDT Key Character Code Transformations

ASCII	Terminal	Terminal Generated		ASCII Task Edit				
Character	Key	Code	Type	Buffer	Event	Flag	ASCII	Direct
NULL	(CONTROL) 3	00		20	80		00	N/A*
SOH	(CONTROL) A	01		20	81		00	
STX	(CONTROL) B	02		20	82		00	
ETX	(CONTROL) C	03		20	83		00	
EOT	(CONTROL) D	04		20	84		00	
ENQ	(CONTROL) E	05		20	85		00	
ACK	(CONTROL) F	06		20	86		00	
BELL	(CONTROL) G	07		20	87			
BS	(CONTROL) H	08						
HT	(CONTROL) I	09		20	89			
LF	(CONTROL) J	0A		20	8A			
VT	(CONTROL) K	0B		20	8B			
FF	(CONTROL) L	0C		20	8C			
CR	(CONTROL) M	0D		20	8D		00	
CR	RETURN	0D	STR	20	8D		00	
SO	(CONTROL) N	0E		20	8E			
SI	(CONTROL) O	0F		20	8F			
DLE	(CONTROL) P	10						
DC1	(CONTROL) Q	11						
DC2	(CONTROL) R	12		20	20		00	
DC3	(CONTROL) S	13	F	20	93	EOF	00 EOF	
DC4	(CONTROL) T	14		20	94			
NAK	(CONTROL) U	15		20	95			
SYN	(CONTROL) V	16		20	96		00	
ETB	(CONTROL) W	17		20	97		00	
CAN	(CONTROL) X	18		20	98		00	
EM	(CONTROL) Y	19		20	99		00	
SUB	(CONTROL) Z	1A		20	9A		00	
ESC	ESC	1B		20	9B		00	
FS	(CONTROL) ,	1C		20	9C		00	
GS	(CONTROL) +	1D		20	9D		00	
RS	(CONTROL) .	1E		20	9E		00	
US	(CONTROL) /	1F		20	9F		00	
Space	Space bar	20		20	20		20	
!	!	21		21	21		21	
"	"	22		22	22		22	
#	#	23		23	23		23	
$	$	24		24	24		24	
%	%	25		25	25		25	
&	&	26		26	26		26	
'	'	27		27	27		27	
(	(	28		28	28		28	
)	)	29		29	29		29	
*	*	2A		2A	2A		2A	

Note:

* Not applicable, does not interact with keyboard.

Table B-1. 911 VDT Key Character Code Transformations (Continued)

ASCII Character	Terminal Key	Terminal Generated Code	Type	ASCII Task Edit Buffer	Event	Flag	ASCII	Direct
+	+	2B		2B	2B		2B	
,	,	2C		2C	2C		2C	
-	-	2D		2D	2D		2D	
.	.	2E		2E	2E		2E	
/	/	2F		2F	2F		2F	
0	0	30		30	30		30	
1	1	31		31	31		31	
2	2	32		32	32		32	
3	3	33		33	33		33	
4	4	34		34	34		34	
5	5	35		35	35		35	
6	6	36		36	36		36	
7	7	37		37	37		37	
8	8	38		38	38		38	
9	9	39		39	39		39	
:	:	3A		3A	3A		3A	
;	;	3B		3B	3B		3B	
<	<	3C		3C	3C		3C	
=	=	3D		3D	3D		3D	
>	>	3E		3E	3E		3E	
?	?	3F		3F	3F		3F	
@	@	40		40	40		40	
A	A	41		41	41		41	
B	B	42		42	42		42	
C	C	43		43	43		43	
D	D	44		44	44		44	
E	E	45		45	45		45	
F	F	46		46	46		46	
G	G	47		47	47		47	
H	H	48		48	48		48	
I	I	49		49	49		49	
J	J	4A		4A	4A		4A	
K	K	4B		4B	4B		4B	
L	L	4C		4C	4C		4C	
M	M	4D		4D	4D		4D	
N	N	4E		4E	4E		4E	
O	O	4F		4F	4F		4F	
P	P	50		50	50		50	
Q	Q	51		51	51		51	
R	R	52		52	52		52	
S	S	53		53	53		53	
T	T	54		54	54		54	
U	U	55		55	55		55	
V	V	56		56	56		56	
W	W	57		57	57		57	

Note:

* Not applicable, does not interact with keyboard.

Table B-1. 911 VDT Key Character Code Transformations (Continued)

ASCII Character	Terminal Key	Terminal Generated Code	Type	ASCII Task Edit			ASCII	Direct
				Buffer	Event	Flag		
X	X	58		58	58		58	
Y	Y	59		59	59		59	
Z	Z	5A		5A	5A		5A	
[	[	5B		5B	5B		5B	
\	(CONTROL) _	5C		5C	5C		5C	
]	]	5D		5D	5D		5D	
^	^	5E		5E	5E		5E	
_	_	5F		5F	5F		5F	
`	(CONTROL) 9	60		60	60		60	
a	a	61		61	61		61	
b	b	62		62	62		62	
c	c	63		63	63		63	
d	d	64		64	64		64	
e	e	65		65	65		65	
f	f	66		66	66		66	
g	g	67		67	67		67	
h	h	68		68	68		68	
i	i	69		69	69		69	
j	j	6A		6A	6A		6A	
k	k	6B		6B	6B		6B	
l	l	6C		6C	6C		6C	
m	m	6D		6D	6D		6D	
n	n	6E		6E	6E		6E	
o	o	6F		6F	6F		6F	
p	p	70		70	70		70	
q	q	71		71	71		71	
r	r	72		72	72		72	
s	s	73		73	73		73	
t	t	74		74	74		74	
u	u	75		75	75		75	
v	v	76		76	76		76	
w	w	77		77	77		77	
x	x	78		78	78		78	
y	y	79		79	79		79	
z	z	7A		7A	7A		7A	
{	(CONTROL) ;	7B		7B	7B		7B	
|	(CONTROL) 8	7C		7C	7C		7C	
}	(CONTROL) '	7D		7D	7D		7D	
~	(CONTROL) 0	7E		7E	7E		7E	
DEL	(CONTROL) -	7F						
	ERASE FIELD	80	S					
	ERASE INPUT	81	ST	20	8E			
	HOME	82	ST	20	8C			
	TAB	83	ST	20	89			

Note:

* Not applicable, does not interact with keyboard.

Table B-1. 911 VDT Key Character Code Transformations (Continued)

ASCII Character	Terminal Key	Terminal Generated Code	Type	ASCII Task Edit Buffer	ASCII Task Edit Event	ASCII Task Edit Flag	ASCII	Direct
	DEL CHAR	84	S					
	SKIP	85	ST	20	8B			
	INS CHAR	86	S					
	FIELD left	87	T	20	94			
	Left arrow	88	S					
	Up arrow	89	T	20	95			
	Right arrow	8A	S	20	20		20	
	Down arrow	8B	T	20	8A			
	FIELD right	8C	T	20	87			
	(CONTROL) 5	8D	E	20	9D	EVT		
	(CONTROL) 6	8E	E	20	9E	EVT		
	(CONTROL) 7	8F	E	20	9F	EVT		
	(CONTROL) 1	90	E	20	80	EVT		
	(CONTROL) 2	91	E	20	9A	EVT		
	F1	92	E	20	81	EVT		
	F2	93	E	20	82	EVT		
	F3	94	E	20	83	EVT		
	F4	95	E	20	84	EVT		
	F5	96	E	20	85	EVT		
	F6	97	E	20	86	EVT		
	F7	98	E	20	96	EVT		
	F8	99	E	20	97	EVT		
	PRINT	9A	E	20	99	EVT		
	CMD	9B	E	20	98	EVT		
	Blank orange	9C	H					
	Blank gray	9F	T	20	8F			
	ENTER	A0	TF	00 EOF	93	EOF		
	(CONTROL) 4	A1	EF	20	9C	EVT		

Note:

* Not applicable, does not interact with keyboard.

Table B-2. 931 VDT Key Character Code Transformations

ASCII Character	Terminal Key	Terminal Generated Code	Type	ASCII Task Edit Buffer	Event	Flag	ASCII	Direct
NULL	(ALT) 1	00						
SOH	(CTRL) A	01						N/A[1]
STX	(CTRL) B	02						
ETX	(CTRL) C	03						
EOT	(CTRL) D	04						
ENQ	(CTRL) E	05						
ACK	(CTRL) F	06						
BELL	(CTRL) G	07						
BS	BACKSPACE	08						
BS	(CTRL) H	08						
HT	TAB	09		20	89			
HT	(CTRL) I	09		20	89		00	
LF	(CTRL) J	0A		20	87		00	
VT	(CTRL) K	0B						
FF	(CTRL) L	0C						
CR	(CTRL) M	0D		20	8D		00	
CR	RETURN	0D		20	8D		00	
CR	ENTER	0D	F	20	93	EOF	00 EOF	
SO	(CTRL) N	0E						
SI	(CTRL) O	0F						
DLE	(CTRL) P	10						
DC1	(CTRL) Q	11[2]						
DC2	(CTRL) R	12						
DC3	(CTRL) S	13[2]	H					
DC4	(CTRL) T	14						
NAK	(CTRL) U	15						
SYN	(CTRL) V	16						
ETB	(CTRL) W	17						
CAN	(CTRL) X	18		20	98		00	
EM	(CTRL) Y	19						
SUB	(CTRL) Z	1A						
ESC	ESC	1B						
ESC	(CTRL) [	1B						
FS	(CTRL) ,	1C						
GS	(CTRL)]	1D						
RS	(CTRL) .	1E						
US	(CTRL) /	1F						
Space	Space bar	20		20	20		20	
!	!	21		21	21		21	
"	"	22		22	22		22	
#	#	23		23	23		23	
$	$	24		24	24		24	
%	%	25		25	25		25	

Notes:

[1] Not applicapable, does not interact with the keyboard.

[2] The DSR recognizes DC1 and DC3 but it never returns them to the application.

Table B-2. 931 VDT Key Character Code Transformations (Continued)

ASCII Character	Terminal Key	Terminal Generated Code	Type	ASCII Task Edit Buffer			ASCII	Direct
				Buffer	Event	Flag		
&	&	26		26	26		26	
'	'	27		27	27		27	
(	(	28		28	28		28	
)	)	29		29	29		29	
#	#	2A		2A	2A		2A	
+	+	2B		2B	2B		2B	
,	,	2C		2C	2C		2C	
-	-	2D		2D	2D		2D	
.	.	2E		2E	2E		2E	
/	/	2F		2F	2F		2F	
0	0	30		30	30		30	
1	1	31		31	31		31	
2	2	32		32	32		32	
3	3	33		33	33		33	
4	4	34		34	34		34	
5	5	35		35	35		35	
6	6	36		36	36		36	
7	7	37		37	37		37	
8	8	38		38	38		38	
9	9	39		39	39		39	
:	:	3A		3A	3A		3A	
;	;	3B		3B	3B		3B	
<	<	3C		3C	3C		3C	
=	=	3D		3D	3D		3D	
>	>	3E		3E	3E		3E	
?	?	3F		3F	3F		3F	
@	@	40		40	40		40	
A	A	41		41	41		41	
B	B	42		42	42		42	
C	C	43		43	43		43	
D	D	44		44	44		44	
E	E	45		45	45		45	
F	F	46		46	46		46	
G	G	47		47	47		47	
H	H	48		48	48		48	
I	I	49		49	49		49	
J	J	4A		4A	4A		4A	
K	K	4B		4B	4B		4B	
L	L	4C		4C	4C		4C	
M	M	4D		4D	4D		4D	
N	N	4E		4E	4E		4E	
O	O	4F		4F	4F		4F	
P	P	50		50	50		50	
Q	Q	51		51	51		51	

Notes:

[1] Not applicapable, does not interact with the keyboard.

[2] The DSR recognizes DC1 and DC3 but it never returns them to the application.

Table B-2. 931 VDT Key Character Code Transformations (Continued)

		Terminal						
ASCII	Terminal	Generated		ASCII Task Edit				
Character	Key	Code	Type	Buffer	Event	Flag	ASCII	Direct
R	R	52		52	52		52	
S	S	53		53	53		53	
T	T	54		54	54		54	
U	U	55		55	55		55	
V	V	56		56	56		56	
W	W	57		57	57		57	
X	X	58		58	58		58	
Y	Y	59		59	59		59	
Z	Z	5A		5A	5A		5A	
[	[	5B		5B	5B		5B	
		5C		5C	5C		5C	
]	]	5D		5D	5D		5D	
^	^	5E		5E	5E		5E	
_	_	5F		5F	5F		5F	
`	`	60		60	60		60	
a	a	61		61	61		61	
b	b	62		62	62		62	
c	c	63		63	63		63	
d	d	64		64	64		64	
e	e	65		65	65		65	
f	f	66		66	66		66	
g	g	67		67	67		67	
h	h	68		68	68		68	
i	i	69		69	69		69	
j	j	6A		6A	6A		6A	
k	k	6B		6B	6B		6B	
l	l	6C		6C	6C		6C	
m	m	6D		6D	6D		6D	
n	n	6E		6E	6E		6E	
o	o	6F		6F	6F		6F	
p	p	70		70	70		70	
q	q	71		71	71		71	
r	r	72		72	72		72	
s	s	73		73	73		73	
t	t	74		74	74		74	
u	u	75		75	75		75	
v	v	76		76	76		76	
w	w	77		77	77		77	
x	x	78		78	78		78	
y	y	79		79	79		79	
z	z	7A		7A	7A		7A	
{	{	7B		7B	7B		7B	
|	|	7C		7C	7C		7C	

Notes:

[1] Not applicapable, does not interact with the keyboard.

[2] The DSR recognizes DC1 and DC3 but it never returns them to the application.

Table B-2. 931 VDT Key Character Code Transformations (Continued)

		Terminal						
ASCII	Terminal	Generated		ASCII Task Edit				
Character	Key	Code	Type	Buffer	Event	Flag	ASCII	Direct
}	}	7D		7D	7D		7D	
~	~	7E		7E	7E		7E	
	(SHIFT) TAB	1B32						
	(SHIFT) Blank gray	1B3C						
	ERASE FIELD	1B3D						
	Up arrow	1B41		20	95			
	Down arrow	1B42		20	8A			
	Right arrow	1B43		20	20		20	
	Left arrow	1B44						
	HOME	1B48		20	8C			
	(SHIFT) ERASE FIELD	1B49						
	(SHIFT) ERASE INPUT	1B4A						
	ERASE INPUT	1B4B		20	8E			
	(SHIFT)CMD	1B4C						
	Blank gray	1B4E		20	8F			
	(SHIFT)DEL			S				
	CHAR	1B4F		S				
	INS CHAR	1B50		S				
	DEL CHAR	1B51		S				
	PRINT	1B57		STE	20	99	EVT	
	(SHIFT) Blank			S				
	orange	1B65						
	(SHIFT) ESC	1B66		STE	20	9B	00	
	Blank orange	1B67		S				
	CMD	1B68		SE	20	98	EVT	
	F1	1B6931		E	20	81	EVT	
	F2	1B6932		E	20	82	EVT	
	F3	1B6933		E	20	83	EVT	
	F4	1B6934		E	20	84	EVT	
	F5	1B6935		E	20	85	EVT	
	F6	1B6936		E	20	86	EVT	
	F7	1B6937		E	20	96	EVT	
	F8	1B6938		E	20	97	EVT	
	F9	1B6939		E	20	80	EVT	
	F10	1B693A		E	20	9A	EVT	
	F11	1B693B		E	20	9C	EVT	
	F12	1B693C		E	20	9D	EVT	
	(SHIFT) F1	1B693D		E	20	9E	EVT	
	(SHIFT) F2	1B693E		E	20	9F	EVT	

Notes:

[1] Not applicable, does not interact with the keyboard.

[2] The DSR recognizes DC1 and DC3 but never returns them to the application.

Table B-2. 931 VDT Key Character Code Transformations (Continued)

ASCII Character	Terminal Key	Terminal Generated Code	Type	ASCII Task Edit Buffer	Event	Flag	ASCII	Direct
	(SHIFT) F3	1B693F						
	(SHIFT) F4	1B6940						
	(SHIFT) F5	1B6941						
	(SHIFT) F6	1B6942						
	(SHIFT) F7	1B6943						
	(SHIFT) F8	1B6944						
	(SHIFT) F9	1B6945						
	(SHIFT) F10	1B6946						
	(SHIFT) F11	1B6947						
	(SHIFT) F12	1B6948						
	FIELD right	1B696F		ST	20	87		
	SKIP	1B73		ST	20	8B		
	FIELD left	1B74		ST	20	94		

Notes:

[1] Not applicapable, does not interact with the keyboard.
[2] The DSR recognizes DC1 and DC3 but it never returns them to the application.

Table B-3. 940 VDT Key Character Code Transformations

ASCII Character	Terminal Key	Terminal Generated Code	Type	ASCII Task Edit Buffer	Event	Flag	ASCII	Direct
NULL	(CTRL)	00						N/A[1]
SOH	(CTRL) A	01						
STX	(CTRL) B	02						
ETX	(CTRL) C	03						
EOT	(CTRL) D	04						
ENQ	(CTRL) E	05						
ACK	(CTRL) F	06						
BELL	(CTRL) G	07						
BS	BACK SPACE	08						
BS	(CTRL) H	08						
HT	TAB right	09		20	89		00	
HT	(CTRL) I	09		20	89		00	
LF	LINE FEED	0A		20	87			
LF	(CTRL) J	0A		20	87			
VT	(CTRL) K	0B						

Notes:

[1] Not applicable, does not interact with the keyboard.
[2] The DSR recognizes DC1 and DC3 but never returns them to the application.

Table B-3. 940 VDT Key Character Code Transformations (Continued)

		Terminal						
ASCII	Terminal	Generated		ASCII Task Edit				
Character	Key	Code	Type	Buffer	Event	Flag	ASCII	Direct
FF	(CTRL) L	0C						
CR	(CTRL) M	0D		20	8D		00	
CR	RETURN	0D		20	8D		00	
SO	(CTRL) N	0E						
SI	(CTRL) O	0F						
DLE	(CTRL) P	10						
DC1	(CTRL) Q	11[2]						
DC2	(CTRL) R	12						
DC3	(CTRL) S	13[2]	H					
DC4	(CTRL) T	14						
NAK	(CTRL) U	15						
SYN	(CTRL) V	16						
ETB	(CTRL) W	17						
CAN	(CTRL) X	18		20	98		00	
EM	(CTRL) Y	19						
SUB	(CTRL) Z	1A						
ESC	ESC	1B						
ESC	(CTRL) [	1B						
FS	(CTRL) \	1C						
GS	(CTRL)]	1D						
RS	SHIFT/(CTRL)^	1E						
US	SHIFT/(CTRL)_	1F						
Space	Space bar	20		20	20		20	
!	!	21		21	21		21	
”	”	22		22	22		22	
#	#	23		23	23		23	
$	$	24		24	24		24	
%	%	25		25	25		25	
&	&	26		26	26		26	
’	’	27		27	27		27	
(	(	28		28	28		28	
)	)	29		29	29		29	
*	*	2A		2A	2A		2A	
+	+	2B		2B	2B		2B	
,	,	2C		2C	2C		2C	
–	–	2D		2D	2D		2D	
.	.	2E		2E	2E		2E	
/	/	2F		2F	2F		2F	
0	0	30		30	30		30	
1	1	31		31	31		31	
2	2	32		32	32		32	
3	3	33		33	33		33	
4	4	34		34	34		34	

Notes:

[1] Not applicable, does not interact with the keyboard.

[2] The DSR recognizes DC1 and DC3 but never returns them to the application.

Table B-3. 940 VDT Key Character Code Transformations (Continued)

ASCII Character	Terminal Key	Terminal Generated Code	Type	ASCII Task Edit Buffer	ASCII Task Edit Event	ASCII Task Edit Flag	ASCII	Direct
5	5	35		35	35		35	
6	6	36		36	36		36	
7	7	37		37	37		37	
8	8	38		38	38		38	
9	9	39		39	39		39	
:	:	3A		3A	3A		3A	
;	;	3B		3B	3B		3B	
<	<	3C		3C	3C		3C	
=	=	3D		3D	3D		3D	
>	>	3E		3E	3E		3E	
?	?	3F		3F	3F		3F	
@	@	40		40	40		40	
A	A	41		41	41		41	
B	B	42		42	42		42	
C	C	43		43	43		43	
D	D	44		44	44		44	
E	E	45		45	45		45	
F	F	46		46	46		46	
G	G	47		47	47		47	
H	H	48		48	48		48	
I	I	49		49	49		49	
J	J	4A		4A	4A		4A	
K	K	4B		4B	4B		4B	
L	L	4C		4C	4C		4C	
M	M	4D		4D	4D		4D	
N	N	4E		4E	4E		4E	
O	O	4F		4F	4F		4F	
P	P	50		50	50		50	
Q	Q	51		51	51		51	
R	R	52		52	52		52	
S	S	53		53	53		53	
T	T	54		54	54		54	
U	U	55		55	55		55	
V	V	56		56	56		56	
W	W	57		57	57		57	
X	X	58		58	58		58	
Y	Y	59		59	59		59	
Z	Z	5A		5A	5A		5A	
[	[	5B		5B	5B		5B	
\	\	5C		5C	5C		5C	
]	]	5D		5D	5D		5D	
^	^	5E		5E	5E		5E	

Notes:

[1] Not applicable, does not interact with the keyboard.

[2] The DSR recognizes DC1 and DC3 but never returns them to the application.

Table B-3. 940 VDT Key Character Code Transformations (Continued)

ASCII Character	Terminal Key	Terminal Generated Code	Type	ASCII Task Edit Buffer	Event	Flag	ASCII	Direct
_	_	5F		5F	5F		5F	
		60		60	60		60	
a	a	61		61	61		61	
b	b	62		62	62		62	
c	c	63		63	63		63	
d	d	64		64	64		64	
e	e	65		65	65		65	
f	f	66		66	66		66	
g	g	67		67	67		67	
h	h	68		68	68		68	
i	i	69		69	69		69	
j	j	6A		6A	6A		6A	
k	k	6B		6B	6B		6B	
l	l	6C		6C	6C		6C	
m	m	6D		6D	6D		6D	
n	n	6E		6E	6E		6E	
o	o	6F		6F	6F		6F	
p	p	70		70	70		70	
q	q	71		71	71		71	
r	r	72		72	72		72	
s	s	73		73	73		73	
t	t	74		74	74		74	
u	u	75		75	75		75	
v	v	76		76	76		76	
w	w	77		77	77		77	
x	x	78		78	78		78	
y	y	79		79	79		79	
z	z	7A		7A	7A		7A	
{	{	7B		7B	7B		7B	
|	|	7C		7C	7C		7C	
}	}	7D		7D	7D		7D	
~	~	7E		7E	7E		7E	
	TAB left	1B32						
	ERASE EOS	1B3D						
	Up arrow	1B41		20	95			
	Down arrow	1B42		20	8A			
	Right arrow	1B43		20	20		20	
	Left arrow	1B44						
	HOME	1B48		20	8C			
	ERASE MSG	1B49						
	ERASE ALL	1B4B		20	8E			
	INS LINE	1B50	S					

Notes:

[1] Not applicapable, does not interact with the keyboard.

[2] The DSR recognizes DC1 and DC3 but it never returns them to the application.

Table B-3. 940 VDT Key Character Code Transformations (Continued)

ASCII Character	Terminal Key	Terminal Generated Code	Type	ASCII Task Edit Buffer	Event	Flag	ASCII	Direct
	DEL LINE	1B51	S					
	PRINT	1B57	STE	20	99	EVT		
	SCROLL DOWN	1B61						
	PREV REGN	1B63						
	NEXT REGN	1B64						
	PREV PAGE	1B65						
	NEXT PAGE	1B66	STE	20	9B		00	
	PREV FORM	1B67	S					
	NEXT FORM	1B68	S					
	F1	1B6931	E	20	81	EVT		
	F2	1B6932	E	20	82	EVT		
	F3	1B6933	E	20	83	EVT		
	F4	1B6934	E	20	84	EVT		
	F5	1B6935	E	20	85	EVT		
	F6	1B6936	E	20	86	EVT		
	F7	1B6937	E	20	96	EVT		
	F8	1B6938	E	20	97	EVT		
	F9	1B6939	E	20	80	EVT		
	F10	1B693A	E	20	9A	EVT		
	F11	1B693B	E	20	9C	EVT		
	F12	1B693C	E	20	9D	EVT		
	F13	1B693D	E	20	9E	EVT		
	F14	1B693E	E	20	9F	EVT		
	F15	1B693F						
	F16	1B6940						
	F17	1B6941						
	F18	1B6942						
	F19	1B6943						
	F20	1B6944						
	F21	1B6945						
	F22	1B6946						
	F23	1B6947						
	F24	1B6948						
	SKIP left	1B74		20	94			
	SKIP right	1B7330		20	8B			
	SEND	1BE9F1	F	20	93	EOF	00 EOF	

Notes:

[1] Not applicable, does not interact with the keyboard.

[2] The DSR recognizes DC1 and DC3 but never returns them to the application.

Table B-4. Business System Terminal Key Character Code Transformations

ASCII Character	Terminal Key	Terminal Generated Code	Type	ASCII Task Edit Buffer	Event	Flag	ASCII	Direct
NULL	(CTRL) ‘	00						N/A[1]
SOH	(CTRL) A	01						
STX	(CTRL) B	02						
ETX	(CTRL) C	03						
EOT	(CTRL) D	04						
ENQ	(CTRL) E	05						
ACK	(CTRL) F	06						
BELL	(CTRL) G	07						
BS	BACK SPACE	08						
BS	(CTRL) H	08						
HT	TAB	09		20	89		00	
HT	(CTRL) I	09		20	89		00	
LF	(CTRL) J	0A		20	87			
VT	(CTRL) K	0B						
FF	(CTRL) L	0C						
CR	(CTRL) M	0D		20	8D		00	
CR	RETURN	0D		20	8D		00	
CR	ENTER	1B6971	F	20	93	EOF	00 EOF	
SO	(CTRL) N	0E						
SI	(CTRL) O	0F						
DLE	(CTRL) P	10						
DC1	(CTRL) Q	11[2]						
DC2	(CTRL) R	12						
DC3	(CTRL) S	13[2]	H					
DC4	(CTRL) T	14						
NAK	(CTRL) U	15						
SYN	(CTRL) V	16						
ETB	(CTRL) W	17						
CAN	(CTRL) X	18		20	98		00	
EM	(CTRL) Y	19						
SUB	(CTRL) Z	1A						
ESC	ESC	1B						
ESC	(CTRL) [	1B						
ESC	(CTRL) {	1B						
FS	(CTRL) \	1C						
GS	CTRL/(SHIFT)]	1D						
RS	CTRL/(SHIFT) 6	1E						
US	CTRL/(SHIFT) _	1F						
Space	Space bar	20		20			20	
!	!	21		21			21	

Notes:

[1] Not applicable, does not interact with keyboard.

[2] The DSR recognizes DC1 and DC3 but never returns them to the application.

[3] The numbers are on the numeric keypad, not across the second row of the keyboard.

Table B-4. Business System Terminal Key Character Code Transformations (Continued)

		Terminal						
ASCII	Terminal	Generated		ASCII Task Edit				
Character	Key	Code	Type	Buffer	Event	Flag	ASCII	Direct
"	"	22		22			22	
#	#	23		23			23	
$	$	24		24			24	
%	%	25		25			25	
&	&	26		26			26	
'	'	27		27			27	
(	(	28		28			28	
)	)	29		29			29	
*	*	2A		2A			2A	
+	+	2B		2B			2B	
,	,	2C		2C			2C	
–	–	2D		2D			2D	
.	.	2E		2E			2E	
/	/	2F		2F			2F	
0	0	30		30			30	
1	1	31		31			31	
2	2	32		32			32	
3	3	33		33			33	
4	4	34		34			34	
5	5	35		35			35	
6	6	36		36			36	
7	7	37		37			37	
8	8	38		38			38	
9	9	39		39			39	
:	:	3A		3A			3A	
;	;	3B		3B			3B	
<	<	3C		3C			3C	
=	=	3D		3D			3D	
>	>	3E		3E			3E	
?	?	3F		3F			3F	
@	@	40		40			40	
A	A	41		41			41	
B	B	42		42			42	
C	C	43		43			43	
D	D	44		44			44	
E	E	45		45			45	
F	F	46		46			46	
G	G	47		47			47	
H	H	48		48			48	
I	I	49		49			49	
J	J	4A		4A			4A	

Notes:

[1] Not applicable, does not interact with keyboard.

[2] The DSR recognizes DC1 and DC3 but never returns them to the application.

[3] The numbers are on the numeric keypad, not across the second row of the keyboard.

Table B-4. Business System Terminal Key Character Code Transformations (Continued)

		Terminal						
ASCII	Terminal	Generated		ASCII Task Edit				
Character	Key	Code	Type	Buffer	Event	Flag	ASCII	Direct
K	K	4B		4B			4B	
L	L	4C		4C			4C	
M	M	4D		4D			4D	
N	N	4E		4E			4E	
O	O	4F		4F			4F	
P	P	50		50			50	
Q	Q	51		51			51	
R	R	52		52			52	
S	S	53		53			53	
T	T	54		54			54	
U	U	55		55			55	
V	V	56		56			56	
W	W	57		57			57	
X	X	58		58			58	
Y	Y	59		59			59	
Z	Z	5A		5A			5A	
[	[	5B		5B			5B	
\	\	5C		5C			5C	
]	]	5D		5D			5D	
^	^	5E		5E			5E	
_	_	5F		5F			5F	
`	`	60		60			60	
a	a	61		61			61	
b	b	62		62			62	
c	c	63		63			63	
d	d	64		64			64	
e	e	65		65			65	
g	g	67		67			67	
h	h	68		68			68	
i	i	69		69			69	
j	j	6A		6A			6A	
k	k	6B		6B			6B	
l	l	6C		6C			6C	
m	m	6D		6D			6D	
n	n	6E		6E			6E	
o	o	6F		6F			6F	
p	p	70		70			70	
q	q	71		71			71	
r	r	72		72			72	
s	s	73		73			73	
t	t	74		74			74	

Notes:

[1] Not applicable, does not interact with keyboard.

[2] The DSR recognizes DC1 and DC3 but never returns them to the application.

[3] The numbers are on the numeric keypad, not across the second row of the keyboard.

Table B-4. Business System Terminal Key Character Code Transformations (Continued)

		Terminal						
ASCII	Terminal	Generated		ASCII Task Edit				
Character	Key	Code	Type	Buffer	Event	Flag	ASCII	Direct
u	u	75		75			75	
v	v	76		76			76	
w	w	77		77			77	
x	x	78		78			78	
y	y	79		79			79	
z	z	7A		7A			7A	
{	{	7B		7B			7B	
|	|	7C		7C			7C	
}	}	7D		7D			7D	
~	~	7E		7E			7E	
	Left arrow	1B44						
	Up arrow	1B41		20	95			
	Right arrow	1B43		20	20		20	
	Down arrow	1B42		20	8A			
	(ALT) ERASE							
	INPUT	1B61						
	(ALT) ERASE							
	FIELD	1B62						
	FIELD right	0A		20	87			
	BACK SPACE	08						
	SKIP	1B73		20	8B			
	FIELD left	1B74		00	94			
	TAB	09		00	89		00	
	(SHIFT) BACK							
	SPACE	1B32						
	DEL CHAR	1B51	S					
	(SHIFT) ESC	1B66	STE	00	9B		00	
	(SHIFT) Blank		S					
	orange	1B65						
	(ALT) CMD	1B64						
	(ALT) Blank							
	orange	1B63						
	CMD	1B68	SE	00	98	EVT		
	Blank orange	1B67						
	HOME	1B48						
	(SHIFT) HOME			00	8C			
	ERASE	1B48						
	FIELD	1B3D						
	(SHIFT) ERASE							
	INPUT	1B4A						
	(SHIFT) ERASE							

Notes:

[1] Not applicapable, does not interact with the keyboard.

[2] The DSR recognizes DC1 and DC3 but it never returns them to the application.

[3] The numbers are on the numberic kiy pad, not across the second row of the keyboard.

Table B-4. Business System Terminal Key Character Code Transformations (Continued)

ASCII	Terminal	Terminal Generated		ASCII Task Edit				
Character	Key	Code	Type	Buffer	Event	Flag	ASCII	Direct
	FIELD	1B49						
	(SHIFT) Blank gray	1B3C						
	INS CHAR	1B50	S					
	Blank gray	1B4E		00	8F			
	PRINT		STE	00	99	EVT		
	(ALT) 1[3]	A2						
	(ALT) 4[3]	1B7931		31	31		31	
	(ALT) 2[3]	A4						
	(ALT) 5[3]	1B7932		32	32		32	
	(ALT) 3[3]	A6						
	(ALT) 6[3]	1B7933		33	33		33	
	ERASE INPUT	1B4B		4B	4B			
	F1	1B6931	E	20	81	EVT		
	F2	1B6932	E	20	82	EVT		
	F3	1B6933	E	20	83	EVT		
	F4	1B6934	E	20	84	EVT		
	F5	1B6935	E	20	85	EVT		
	F6	1B6936	E	20	86	EVT		
	F7	1B6937	E	20	96	EVT		
	F8	1B6938	E	20	97	EVT		
	(SHIFT) F1	1B6939	E	20	80	EVT		
	(SHIFT) F2	1B693A	E	20	9A	EVT		
	(SHIFT) F3	1B693B	E	20	9C	EVT		
	(SHIFT) F4	1B693C	E	20	9D	EVT		
	(SHIFT) F5	1B693D	E	20	9E	EVT		
	(SHIFT) F6	1B693E	E	20	9F	EVT		

Notes:

[1] Not applicable, does not interact with keyboard.

[2] The DSR recognizes DC1 and DC3 but never returns them to the application.

[3] The numbers are on the numeric keypad, not across the second row of the keyboard.

Table B-5. 820 KSR Key Character Code Transformations

		Terminal						
ASCII	Terminal	Generated		ASCII Task Edit				
Character	Key	Code	Type	Buffer	Event	Flag	ASCII	Direct
NULL	(CTRL) [	00/80	ES	00	80	EVT		00
SOH	(CTRL) A	81/01	EST	00	81	EVT		01
STX	(CTRL) B	82/02	EST	00	82	EVT		02
ETX	(CTRL) C	03/83	EST	00	83	EVT		03
EOT	(CTRL) D	84/04	ES	00	84	EVT		04
ENQ	(CTRL) E	05/85	EST	00	85	EVT		05
ACK	(CTRL) F	06/86	ES	00	86	EVT		06
BELL	(CTRL) G	87/07						07
BS	(CTRL) H	88/08						08
BS	(BACKSPACE	88/08						08
HT	(CTRL) I	09/89	ST	00	89		09	09
HT	TAB	09/89	ST	00	89		09	09
LF	(CTRL) J	0A/8A	S	00	8A		0A	0A
LF	LINE FEED	0A/8A	S	00	8A		0A	0A
VT	(CTRL) K	8B/0B	ST	00	8B			0B
FF	(CTRL) L	0C/8C	ST	00	8C			0C
CR	(CTRL) M	8D/0D	S	00	8D		00	0D
CR	RETURN	8D/0D	S	00	8D		00	0D
SO	(CTRL) N	8E/0E	EST	00	8E			0E
SI	(CTRL) O	0F/8F	E	00	8F			0F
DLE	(CTRL) P	90/10						10
DC1	(CTRL) Q	11/91						11
DC2	(CTRL) R	12/92						12
DC3	(CTRL) S	93/13	H					
DC4	(CTRL) T	14/94	T	00	94			14
NAK	(CTRL) U	95/15	T	00	95			15
SYN	(CTRL) V	96/16	E	00	96	EVT		16
ETB	(CTRL) W	17/97	E	00	97	EVT		17
CAN	(CTRL) X	18/98	E	00	98	EVT		18
EM	(CTRL) Y	99/19	EF	00	93	EVT EOF	00 EOF	00
SUB	(CTRL) Z	9A/1A	E	00	9A	EVT		1A
ESC	ESC	1B/9B						1B
FS	(CTRL) \	9C/1C	E	00	9C	EVT		1C
GS	(CTRL) {	1D/9D	E	00	9D	EVT		1D
RS	(CTRL) =	1E/9E	E	00	9E	EVT		1E
US	(CTRL) -	9F/1F	ET	00	9F	EVT		1F
Space	Space bar	A0/20		20			20	20
!	!	21/A1		21			21	21
"	"	22/A2		22			22	22
#	#	3/23		23			23	23
$	$	24/A4		24			24	24
%	%	A5/25		25			25	25
&	&	A6/26		26			26	26
'	'	27/A7		27			27	27
(	(	28/A8		28			28	28
)	)	A9/29		29			29	29
*	*	AA/2A		2A			2A	2A
+	+	2B/AB		2B			2B	2B

Table B-5. 820 KSR Key Character Code Transformations (Continued)

ASCII Character	Terminal Key	Terminal Generated Code	Type	ASCII Task Edit Buffer	Event	Flag	ASCII	Direct
,	,	AC/2C		2C			2C	2C
-	-	2D/AD		2D			2D	2D
.	.	2E/AE		2E			2E	2E
/	/	AF/2F		2F			2F	2F
0	0	30/B0		30			30	30
1	1	B1/31		31			31	31
2	2	B2/32		32			32	32
3	3	33/B3		33			33	33
4	4	B4/34		34			34	34
5	5	35/B5		35			35	35
6	6	36/B6		36			36	36
7	7	B7/37		37			37	37
8	8	B8/38		38			38	38
9	9	39/B9		39			39	39
:	:	3A/BA		3A			3A	3A
;	;	BB/3B		3B			3B	3B
<	<	3C/BC		3C			3C	3C
=	=	BD/3D		3D			3D	3D
>	>	BE/3E		3E			3E	3E
?	?	3F/BF		3F			3F	3F
@	@	C0/40		40			40	40
A	A	41/C1		41			41	41
B	B	42/C2		42			42	42
C	C	C3/43		43			43	43
D	D	44/C4		44			44	44
E	E	C5/45		45			45	45
F	F	C6/46		46			46	46
G	G	47/C7		47			47	47
H	H	48/C8		48			48	48
I	I	C9/49		49			49	49
J	J	CA/4A		4A			4A	4A
K	K	4B/CB		4B			4B	4B
L	L	CC/4C		4C			4C	4C
M	M	4D/CD		4D			4D	4D
N	N	4E/CE		4E			4E	4E
O	O	CF/4F		4F			4F	4F
P	P	50/D0		50			50	50
Q	Q	D1/51		51			51	51
R	R	D2/52		52			52	52
S	S	53/D3		53			53	53
T	T	D4/54		54			54	54
U	U	55/D5		55			55	55
V	V	56/D6		56			56	56
W	W	D7/57		57			57	57
X	X	D8/58		58			58	58
Y	Y	59/D9		59			59	59
Z	Z	5A/DA		5A			5A	5A

Table B-5. 820 KSR Key Character Code Transformations (Continued)

		Terminal						
ASCII	Terminal	Generated		ASCII Task Edit				
Character	Key	Code	Type	Buffer	Event	Flag	ASCII	Direct
[	[	DB/5B		5B			5B	5B
\	\	5C/DC		5C			5C	5C
]	]	DD/5D		5D			5D	5D
^	^	DE/5E		5E			5E	5E
_	_	5F/DF		5F			5F	5F
`	`	60/E0		60			60	60
a	a	E1/61		61			61	61
b	b	E2/62		62			62	62
c	c	63/E3		63			63	63
d	d	E4/64		64			64	64
e	e	65/E5		65			65	65
f	f	66/E6		66			66	66
g	g	E7/67		70			67	67
h	h	E8/68		68			68	68
i	i	69/E9		69			69	69
j	j	6A/EA		6A			6A	6A
k	k	EB/6B		6B			6B	6B
l	l	6C/EC		6C			6C	6C
m	m	ED/6D		6D			6D	6D
n	n	EE/6E		6E			6E	6E
o	o	6F/EF		6F			6F	6F
p	p	F0/70		70			70	70
q	q	71/F1		71			71	71
r	r	72/F2		72			72	72
s	s	F3/73		73			73	73
t	t	74/F4		74			74	74
u	u	F5/75		75			75	75
v	v	F6/76		76			76	76
w	w	77/F7		77			77	77
x	x	78/F8		78			78	78
y	y	F9/79		79			79	79
z	z	FA/7A		7A			7A	7A
{	{	7B/FB		7B			7B	7B
|	|	FC/7C		7C			7C	7C
}	}	7D/FD		7D			7D	7D
~	~	7E/FE		7E			7E	7E
DEL	DEL	00/7F						

Table B-6. 783 TPD Key Character Code Transformations

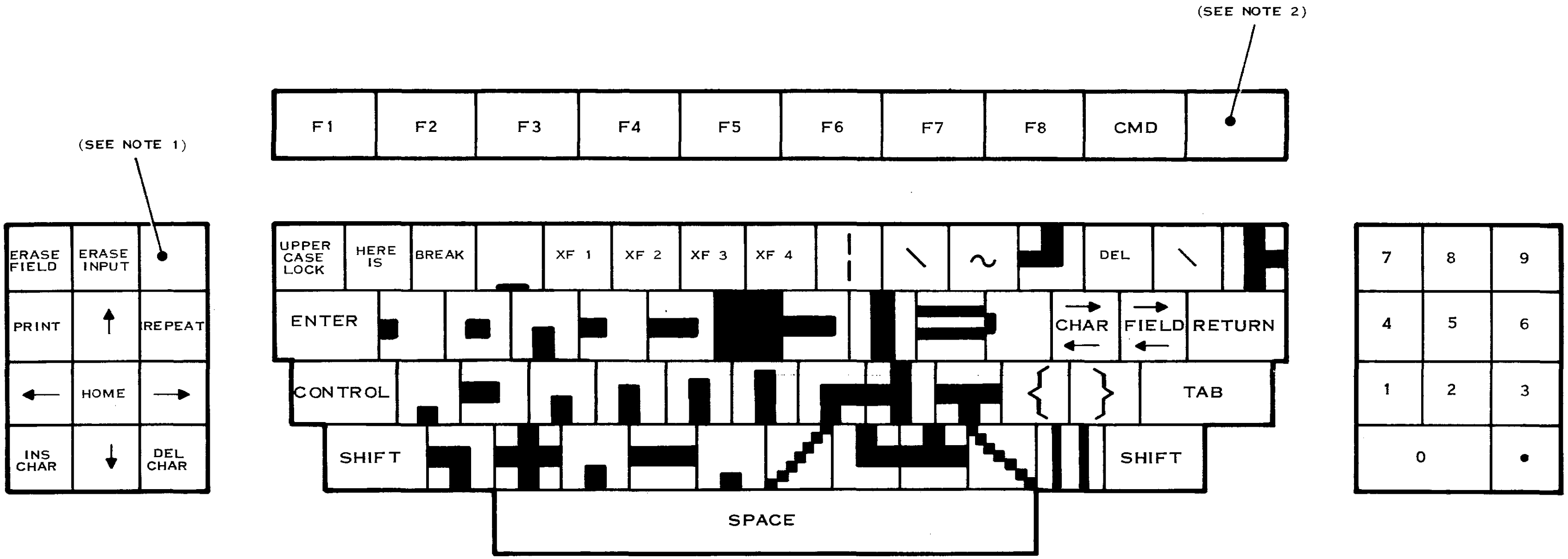
ASCII	Terminal	Terminal Generated		ASCII Task Edit				
Character	Key	Code	Type	Buffer	Event	Flag	ASCII	Direct
NULL	(CTRL) [	00/80	SE	00	80	EVT		00
SOH	(CTRL) A	81/01	STE	00	81	EVT		01
STX	(CTRL) B	82/02	STE	00	82	EVT		02
ETX	(CTRL) C	03/83	STE	00	83	EVT		03
EOT	(CTRL) D	84/04	SE	00	84	EVT		04
ENQ	(CTRL) E	05/85	STE	00	85	EVT		05
ACK	(CTRL) F	06/86	SE	00	86	EVT		06
BELL	(CTRL) G	87/07						07
BS	(CTRL) H	88/08						08
HT	(CTRL) I	09/89	ST	00	89		09	09
LF	(CTRL) J	0A/8A	S	00	8A		0A	0A
LF	LINE FEED	0A/8A	S	00	8A		0A	0A
VT	(CTRL) K	8B/0B	ST	00	8B			0B
FF	(CTRL) L	0C/8C	ST	00	8C			0C
CR	(CTRL) M	8D/0D	S	00	8D		00	0D
CR	RETURN	8D/0D	S	00	8D		00	0D
CR	ENTER	8D/0D	S	00	8D		00	0D
SO	(CTRL) N	8E/0E	STE	00	8E			0E
SI	(CTRL) O	0F/8F	E	00	8F			0F
DLE	(CTRL) P	90/10						10
DC1	(CTRL) Q	11/91						11
DC2	(CTRL) R	12/92						12
DC3	(CTRL) S	93/13	H					
DC4	(CTRL) T	14/94	T	00	94			14
NAK	(CTRL) U	95/15	T	00	95			15
SYN	(CTRL) V	96/16	E	00	96	EVT		16
ETB	(CTRL) W	17/97	E	00	97	EVT		17
CAN	(CTRL) X	18/98	E	00	98	EVT		18
EM	(CTRL) Y	99/19	EF	00	93	EVT EOF	00 EOF	00
SUB	(CTRL) Z	9A/1A	E	00	9A	EVT		1A
ESC	ESC	1B/9B						1B
FS	(CTRL) \	9C/1C	E	00	9C	EVT		1C
GS	(CTRL) {	1D/9D	E	00	9D	EVT		1D
RS	(CTRL) =	1E/9E	E	00	9E	EVT		1E
US	(CTRL) -	9F/1F	ET	00	9F	EVT		1F
Space	Space bar	A0/20		20			20	20
!	!	21/A1		21			21	21
"	"	22/A2		22			22	22
#	#	A3/23		23			23	23
$	$	24/A4		24			24	24
%	%	A5/25		25			25	25
&	&	A6/26		26			26	26
'	'	27/A7		27			27	27
(	(	28/A8		28			28	28
)	)	A9/29		29			29	29
*	*	AA/2A		2A			2A	2A
+	+	2B/AB		2B			2B	2B

Table B-6. 783 TPD Key Character Code Transformations (Continued)

		Terminal						
ASCII	Terminal	Generated		ASCII Task Edit				
Character	Key	Code	Type	Buffer	Event	Flag	ASCII	Direct
,	,	AC/2C		2C			2C	2C
-	-	2D/AD		2D			2D	2D
.	.	2E/AE		2E			2E	2E
/	/	AF/2F		2F			2F	2F
0	0	30/B0		30			30	30
1	1	B1/31		31			31	31
2	2	B2/32		32			32	32
3	3	33/B3		33			33	33
4	4	B4/34		34			34	34
5	5	35/B5		35			35	35
6	6	36/B6		36			36	36
7	7	B7/37		37			37	37
8	8	B8/38		38			38	38
9	9	39/B9		39			39	39
:	:	3A/BA		3A			3A	3A
;	;	BB/3B		3B			3B	3B
<	<	3C/BC		3C			3C	3C
=	=	BD/3D		3D			3D	3D
>	>	BE/3E		3E			3E	3E
?	?	3F/BF		3F			3F	3F
@	@	C0/40		40			40	40
A	A	41/C1		41			41	41
B	B	42/C2		42			42	42
C	C	C3/43		43			43	43
D	D	44/C4		44			44	44
E	E	C5/45		45			45	45
F	F	C6/46		46			46	46
G	G	47/C7		47			47	47
H	H	48/C8		48			48	48
I	I	C9/49		49			49	49
J	J	CA/4A		4A			4A	4A
K	K	4B/CB		4B			4B	4B
L	L	CC/4C		4C			4C	4C
M	M	4D/CD		4D			4D	4D
N	N	4E/CE		4E			4E	4E
O	O	CF/4F		4F			4F	4F
P	P	50/D0		50			50	50
Q	Q	D1/51		51			51	51
R	R	D2/52		52			52	52
S	S	53/D3		53			53	53
T	T	D4/54		54			54	54
U	U	55/D5		55			55	55
V	V	56/D6		56			56	56
W	W	D7/57		57			57	57
X	X	D8/58		58			58	58
Y	Y	59/D9		59			59	59
Z	Z	5A/DA		5A			5A	5A

Table B-6. 783 TPD Key Character Code Transformations (Continued)

		Terminal						
ASCII	Terminal	Generated		ASCII Task Edit				
Character	Key	Code	Type	Buffer	Event	Flag	ASCII	Direct
[	[	DB/5B		5B			5B	5B
\	(CTRL) 7	5C/DC		5C			5C	5C
]	]	DD/5D		5D			5D	5D
^	^	DE/5E		5E			5E	5E
_	_	5F/DF		5F			5F	5F
`	(CTRL) 0	60/E0		60			60	60
a	a	E1/61		61			61	61
b	b	E2/62		62			62	62
c	c	63/E3		63			63	63
d	d	E4/64		64			64	64
e	e	65/E5		65			65	65
f	f	66/E6		66			66	66
g	g	E7/67		67			67	67
h	h	E8/68		68			68	68
i	i	69/E9		69			69	69
j	j	6A/EA		6A			6A	6A
k	k	EB/6B		6B			6B	6B
l	l	6C/EC		6C			6C	6C
m	m	ED/6D		6D			6D	6D
n	n	EE/6E		6E			6E	6E
o	o	6F/EF		6F			6F	6F
p	p	F0/70		70			70	70
q	q	71/F1		71			71	71
r	r	72/F2		72			72	72
s	s	F3/73		73			73	73
t	t	74/F4		74			74	74
u	u	F5/75		75			75	75
v	v	F6/76		76			76	76
w	w	77/F7		77			77	77
x	x	78/F8		78			78	78
y	y	F9/79		79			79	79
z	z	FA/7A		7A			7A	7A
{	{	7B/FB		7B			7B	7B
|	(CTRL) 8	FC/7C		7C			7C	7C
}	}	7D/FD		7D			7D	7D
~	(CTRL) 9	7E/FE		7E			7E	7E
DEL	DEL	00/7F						

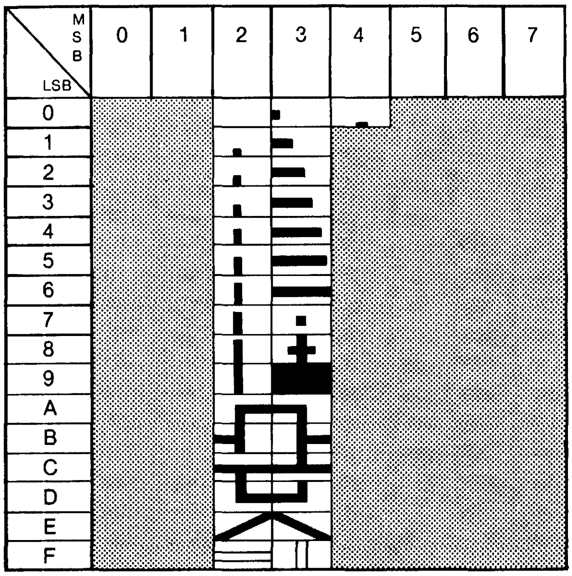


NOTE: 1. GENERATES HEX CODE 9F
2. GENERATES HEX CODE 9C

2283184

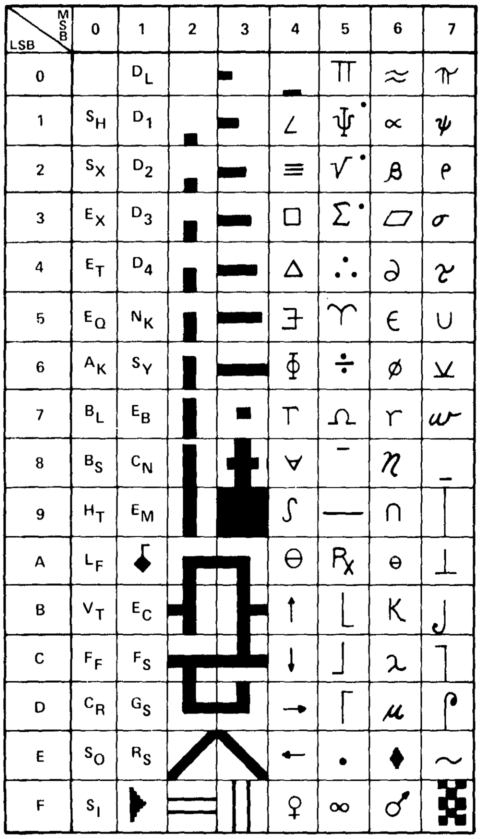
Figure B-1. 911 VDT Graphics Character Keyboard Positions

MSB / LSB	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	@	P	`	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	¦
D	CR	GS	-	=	M	]	m	}
E	SO	RS	°	>	N	^	n	~
F	SI	US	/	?	O	—	o	DEL



2284669

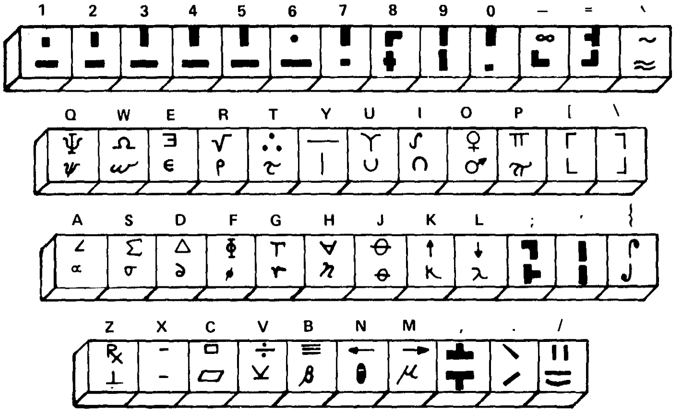
Figure B-2. 931 VDT Graphics Characters



LSB \ MSB	0	1	2	3	4	5	6	7
0		D_L		▬	▁	Π	≈	π
1	S_H	D_1	▪	▬	∠	Ψ•	∝	ψ
2	S_X	D_2	▪	▬	≡	√•	β	ρ
3	E_X	D_3	▮	▬	□	Σ•	▱	σ
4	E_T	D_4	▮	▬	△	∴	∂	τ
5	E_Q	N_K	▮	▬	∃	ϒ	∈	∪
6	A_K	S_Y	▮	▬	Φ	÷	∅	⊻
7	B_L	E_B	▮	▪	T	Ω	γ	ω
8	B_S	C_N	▮	╋	∀	¯	η	_
9	H_T	E_M	▮	█	∫	—	∩	⌉
A	L_F	♦	┏	┓	Θ	℞	θ	⊥
B	V_T	E_C	┫	┣	↑	└	K	⌡
C	F_F	F_S	╋	╋	↓	┘	λ	┐
D	C_R	G_S	┗	┛	→	┌	μ	⌠
E	S_O	R_S	╱	╲	←	•	♦	~
F	S_I	▶	═	║	♀	∞	♂	▚

SC2 Character Set

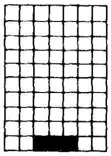
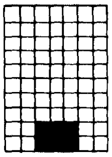
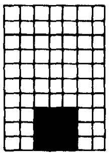
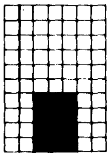
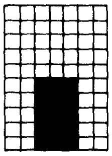
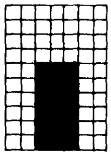
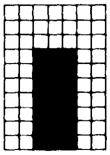
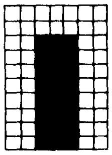
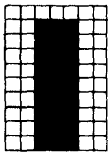
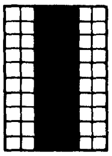
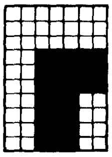
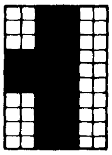
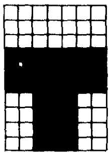
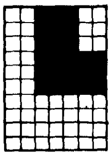
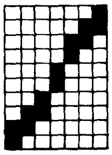
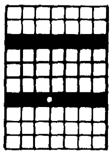
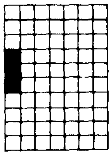
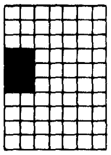
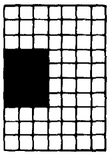
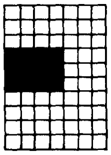
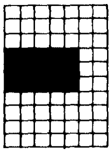
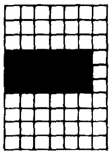
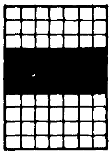
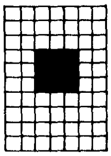
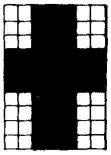
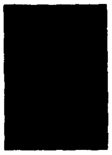
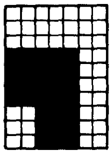
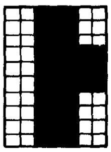
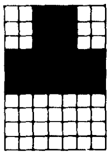
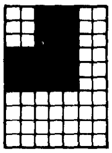
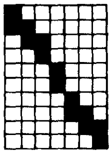
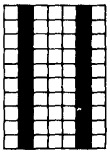
2280966



SC2 Keyboard Layout

Figure B-3. 940 VDT Graphics Characters

Table B-7. Model 911 VDT Graphics Character Sets

CODE	CHARACTER	CODE	CHARACTER	CODE	CHARACTER	CODE	CHARACTER
00	 *	01	 *	02	 *	03	 *
04	 *	05	 *	06	 *	07	 *
08	 *	09	 *	0A	 *	0B	 *
0C	 *	0D	 *	0E	 *	0F	 *
10	 *	11	 *	12	 *	13	 *
14	 *	15	 *	16	 *	17	 *
18	 *	19	 *	1A	 *	1B	 *
1C	 *	1D	 *	1E	 *	1F	 *

2279759

B.3 733 AND 743 DATA TERMINALS

Table B-8 lists the character set of the 733 and 743 data terminals (when they are included in system generation as ASR or KSR device types) followed by an explanation of the characters that perform special functions on these terminals.

Table B-8. 733 and 743 Data Terminal Character Set

Character	Hexadecimal	Character	Hexadecimal	Character	Hexadecimal
Space	20	@	40	’	60
!	21	A	41	a	61
“	22	B	42	b	62
#	23	C	43	c	63
$	24	D	44	d	64
%	25	E	45	e	65
&	26	F	46	f	66
’	27	G	47	g	67
(	28	H	48	h	68
)	29	I	49	i	69
*	2A	J	4A	j	6A
+	2B	K	4B	k	6B
,	2C	L	4C	l	6C
-	2D	M	4D	m	6D
.	2E	N	4E	n	6E
/	2F	O	4F	o	6F
0	30	P	50	p	70
1	31	Q	51	q	71
2	32	R	52	r	72
3	33	S	53	s	73
4	34	T	54	t	74
5	35	U	55	u	75
6	36	V	56	v	76
7	37	W	57	w	77
8	38	X	58	x	78
9	39	Y	59	y	79
:	3A	Z	5A	z	7A
;	3B	[	5B	{	7B
<	3C	/	5C	|	7C
=	3D	]	5D	}	7D
>	3E	^	5E	~	7E
?	3F	_	5F		

Table B-8. 733 and 743 Data Terminal Character Set (Continued)

Character	Hexadecimal	Character	Hexadecimal	Character	Hexadecimal

Notes:

Keyboard:

1. BS character (>08) is returned to printer as LF and BS. Deletes most recently entered character in buffer.
2. HT character (>09) is returned to the printer as space. Character is placed in buffer.
3. LF character (>0A) is returned to printer as LF, but is not stored in buffer.
4. CR character (>0D) is returned to printer as CR, and character is not placed in buffer. Character terminates the record.
5. DC3 character (>13) is not stored in buffer. When DC3 is first character of record, the record is an end-of-file record.
6. ESC character (>1B), when entered during output, terminates output with a write error.
7. DEL character (>7F) is returned to printer as a line feed and carriage return. Character deletes current input record.
8. Maximum buffer size is 83 characters.

Printer:

1. BS character (>08) results in a backspace operation.
2. HT character (>09) results in printing a space.
3. LF character (>0A) results in a line feed operation.
4. CR character (>0D) results in a carriage return operation.
5. End-of-record occurs when specified number of characters have been printed.
6. Maximum buffer size is 83 characters.

Cassette Input:

1. The characters LF (>0A) and DEL (>7F) or the character DEL at the beginning of a record are ignored. The first valid character of the record is the character following the DEL.
2. The characters HT (>09), FF (>0C), BEL (>07), and BS (>08) are stored in the user's buffer.
3. The character ETB (>17) is stored in the user's buffer as a CR (.0D).
4. The character DC3 (>13) as the first character of a record indicates an end-of-file record. The system returns end-of-file status after positioning the tape at the beginning of the next record. The DC3 is not stored in the buffer.
   An EOF may be added to a cassette by the following procedure:
   a. Position the cassette for recording the EOF record.
   b. With the cassette drive in the local record mode, type the character sequence DC3 (CTRL S), RETURN, and NEW LINE on the keyboard.
   c. Press the RECORD OFF button.
   d. Rewind the cassette.
5. The character CR (>0D) indicates end-of-record, and is not stored in the buffer.
6. Maximum buffer size is 83 characters for U.S. and European terminals, and 80 characters for Katakana terminals.

Cassette Output:

1. The end-of-block character sequence is CR (>0D) LF (>0A) DC4(>14) DEL (>7F). The end-of-file character sequence is DC3 (>13) CR DC4 DEL. These characters are supplied by the system, not by the user.
2. The characters HT (>09), FF (>0C), BEL (>07), and BS (>08) are written unchanged.
3. The character CR (>0D) is translated to ETB (>17) and written.
4. The character DC3 (>13) may be placed within a record, but may not be the first character of a record other than the end-of-file record.
5. End-of-record occurs when specified number of characters have been written.
6. Maximum buffer size is 83 characters for U.S. and European terminals, and 80 characters for Katakana terminals.

B.3.1 End-of-File Sequence

The end-of-file sequence for the 733 and 743 data terminals is a DC3 character.

B.3.2 End-of-Record Sequence

The end-of-record sequence for the 733 and 743 data terminals is a CR character.

B.4 CARD READER

Table B-9 provides a hexadecimal to row punch conversion table.

Table B-9. Card Reader Character Set

Hexadecimal	Row Punches	Hexadecimal	Row Punches
00	12-0-9-8-1	2F	0-1
01	12-9-1	30	0
02	12-9-2	31	1
03	12-9-3	32	2
04	9-7	33	3
05	0-9-8-5	34	4
06	0-9-8-6	35	5
07	0-9-8-7	36	6
08	11-9-6	37	7
09	12-9-5	38	8
0A	0-9-5	39	9
0B	12-9-8-3	3A	8-2
0C	12-9-8-4	3B	11-8-6
0D	12-9-8-5	3C	12-8-4
0E	12-9-8-6	3D	8-6
0F	12-9-8-7	3E	0-8-6
10	12-11-9-8-1	3F	0-8-7
11	11-9-1	40	8-4
12	11-9-2	41	12-1
13	11-9-3	42	12-2
14	9-8-4	43	12-3
15	9-8-5	44	12-4
16	9-2	45	12-5
17	0-9-6	46	12-6
18	11-9-8	47	12-7
19	11-9-8-1	48	12-8
1A	9-8-7	49	12-9
1B	0-9-7	4A	11-1
1C	11-9-8-4	4B	11-2
1D	11-9-8-5	4C	11-3
1E	11-9-8-6	4D	11-4
1F	11-9-8-7	4E	11-5
20	None	4F	11-6
21	12-8-7	50	11-7
22	8-7	51	11-8
23	8-3	52	11-9

Table B-9. Card Reader Character Set (Continued)

Hexadecimal	Row Punches	Hexadecimal	Row Punches
24	11-8-3	53	0-2
25	0-8-4	54	0-3
26	12	55	0-4
27	8-5	56	0-5
28	12-8-5	57	0-6
29	11-8-5	58	0-7
2A	11-8-4	59	0-8
2B	12-8-6	5A	0-9
2C	0-8-3	5B	12-8-2
2D	11	5C	0-8-2
2E	12-8-3	5D	11-8-2
5E	11-8-7	6F	12-11-6
5F	0-8-5	70	12-11-7
60	8-1	71	12-11-8
61	12-0-1	72	12-11-9
62	12-0-2	73	11-0-2
63	12-0-3	74	11-0-3
64	12-0-4	75	11-0-4
65	12-0-5	76	11-0-5
66	12-0-6	77	11-0-6
67	12-0-7	78	11-0-7
68	12-0-8	79	11-0-8
69	12-0-9	7A	11-0-9
6A	12-11-1	7B	12-0
6B	12-11-2	7C	12-11
6C	12-11-3	7D	11-0
6D	12-11-4	7E	11-0-1
6E	12-11-5	7F	12-9-7

Notes:

End of record occurs when the specified number of characters, or 80 characters have been read.

Maximum buffer size is 80 characters.

B.4.1 End-of-File Sequence

The end-of-file sequence for the card reader is a slash (/) in column one and an asterisk (*) in column two.

B.5 LINE PRINTER

DNOS provides support for various types of line printers, including line printers supporting the 8-bit Katakana code. The characters which have special significance for the line printer are listed in Table B-10.

Table B-10. Line Printer Character Set

Character	Hexadecimal	Character	Hexadecimal	Character	Hexadecimal
Space	20	@	40	’	60
!	21	A	41	a	61
“	22	B	42	b	62
#	23	C	43	c	63
$	24	D	44	d	64
%	25	E	45	e	65
&	26	F	46	f	66
,	27	G	47	g	67
(	28	H	48	h	68
)	29	I	49	i	69
*	2A	J	4A	j	6A
+	2B	K	4B	k	6B
,	2C	L	4C	l	6C
-	2D	M	4D	m	6D
.	2E	N	4E	n	6E
/	2F	O	4F	o	6F
0	30	P	50	p	70
1	31	Q	51	q	71
2	32	R	52	r	72
3	33	S	53	s	73
4	34	T	54	t	74
5	35	U	55	u	75
6	36	V	56	v	76
7	37	W	57	w	77
8	38	X	58	x	78
9	39	Y	59	y	79
:	3A	Z	5A	z	7A
;	3B	[	5B	{	7B
<	3C	/	5C	|	7C
=	3D	]	5D	}	7D
>	3E	^	5E	~	7E
?	3F	-	5F	DEL	7F

Notes:

The character BS (> 08) results in a backspace operation.

The character HT (> 08) results in a single space.

The character LF (> 0A) results in a line feed operation.

The character CR (> 0D) results in a carriage return operation.

The character FF (> 0C) results in a form feed operation.

The character BEL (> 07) results in a tone signal.

For Model 306, Model 810, and Model 588 Line Printers, the character S0 (>0E) results in character elongation for the line of characters following the S0. Elongation doubles the width of the characters, and a line of elongated characters contains one-half the number of characters on a normal line; i.e., 40 or 66. Fifty is ignored by 2230 and 2260 line printers.

B.5.1 810 Line Printer Control Characters

The following control characters can be sent using the write direct call:

VT (Vertical Tab) causes data, if any, in the line buffer to be printed and advances the paper to the next vertical tab location or top of form, whichever occurs first. If no vertical tabs are set, a VT command causes the paper to be advanced to top of form.

HT (Horizontal Tab) causes spaces to be entered in the line buffer up to the next horizontal tab location, where printing will begin.

DC1 (Select) selects the printer, enabling it to receive data. The controller responds to this control character but does not transmit a verification.

DC2 + n (Tab to Line) causes the paper drive system to advance to the line specified by n. The value n must be greater than the present line.

DC3 (Does not select) causes the printer to go offline and any subsequent characters transmitted to be discarded until a DC1 character is received. The printer does not go offline until an LF character is received after the DC3; any characters received before the DC3 are printed. The controller responds to this control character but does not transmit a verification.

DEL (Delete) clears the line buffer.

NUL (Null) terminates the tab setting sequence (see below); otherwise it is ignored.

DC4 + n (Tab to Address) causes the carriage to advance at high speed to the column specified by n. The value n must be greater than the present carriage position. If n is less than the present carriage position, this command is ignored.

ESC + 1 + n_1 + n_2 + ... + n_k + NUL (Set Vertical Tabs) clears all existing vertical tabs and sets new tabs at lines n_1, n_2,..., and n_k.

ESC + 2 + n (Set Form Length) sets the form length used by the Form Feed (FF) command to n. Lines per form is set to a default value of 66 at power up.

ESC + 3 + n_1 + n_2 + ... + n_k + NUL (Set Horizontal Tabs) clears existing horizontal tabs and sets new tabs at locations n_1, n_2,..., and n_k.

ESC + 4 sets paper drive system to 6 lines per inch.

ESC + 5 sets paper drive system to 8 lines per inch.

ESC + 6 sets carriage system to 10 characters per inch.

ESC + 7 sets carriage system to 16.5 characters per inch.

ESC + 8 enters parameters into the VFC or VCO channel.

ESC + *9* retrieves parameters from the VFC or VCO channel.

ESC + *;* sets the line length to 132 characters.

ESC + *:* + *n* sets the line length to n characters (where n equals 2 through 126).

NOTE

> The values n, n_1, n_2, and so on, used in the ESC commands represent seven-bit binary numbers. If the parity option is selected on the printer, correct parity must be supplied also.

A horizontal or vertical control character sequence affects lines following the control character sequence only.

B.6 KIF COLLATING SEQUENCE

A description of the collating sequences for the various country codes supported by DNOS appears in Table B-11.

Table B-11. KIF Collating Sequence

France/Belgium	ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz
France Word Processing	ABCDEFGHIJKLMNOPQRSTUVWXYZ aàbcçdeéèfghijklmnopqrstuùvwxyz
Germany/Austria	AÄBCDEFGHIJKLMNOÖPQRSTUÜVWXYZ aäbcdefghijklmnoöpqrsßtuüvwxyz
Japan (Katakana)	ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz ヲァィゥェォャュョッーア イウエオカキクケコサシス セソタチツテトナニヌネノ ハヒフヘホマミムメモヤユ ヨラリルレロワン゛゜
Norway/Denmark	ABCDEFGHIJKLMNOPQRSTUVWXYZ ÆØÅ abcdefghijklmnopqrstuvwxyz æøå
Sweden/Finland	ABCDEÉFGHIJKLMNOPQRSTUVWXYÜZÅÄÖ abcdeéfghijklmnopqrstuvwxyüzåäö
Spanish-speaking countries	ABCDEFGHIJKLMNÑOPQRSTUVWXYZ abcçdefghijklmnñopqrstuvwxyz
Switzerland	ABCDEFGHIJKLMNOPQRSTUVWXYZ aàäbcçdeéèfghijklmnoopqrstuüvwxyz
United Kingdom	ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz

2283037

Appendix C

Master/Slave Task Examples

An example illustrating the typical sequence of master/slave channel operations performed by the master task to process a request from a slave task is shown in Figure C-1. The request from the slave task is shown in Figure C-2.

```
           SDSMAC              16:11:02 TUESDAY, JUN 15, 1982.
 ACCESS NAMES TABLE                                                 PAGE 0001

 SOURCE ACCESS NAME=      VOL1.IPC.IPCMSM
 OBJECT ACCESS NAME=      VOL1.IPC.IPCMSMO
 LISTING ACCESS NAME=     VOL1.IPC.IPCMSML
 ERROR ACCESS NAME=
 OPTIONS=                 XREF,TUNLST,BUNLST,DUNLST
 MACRO LIBRARY PATHNAME=
```

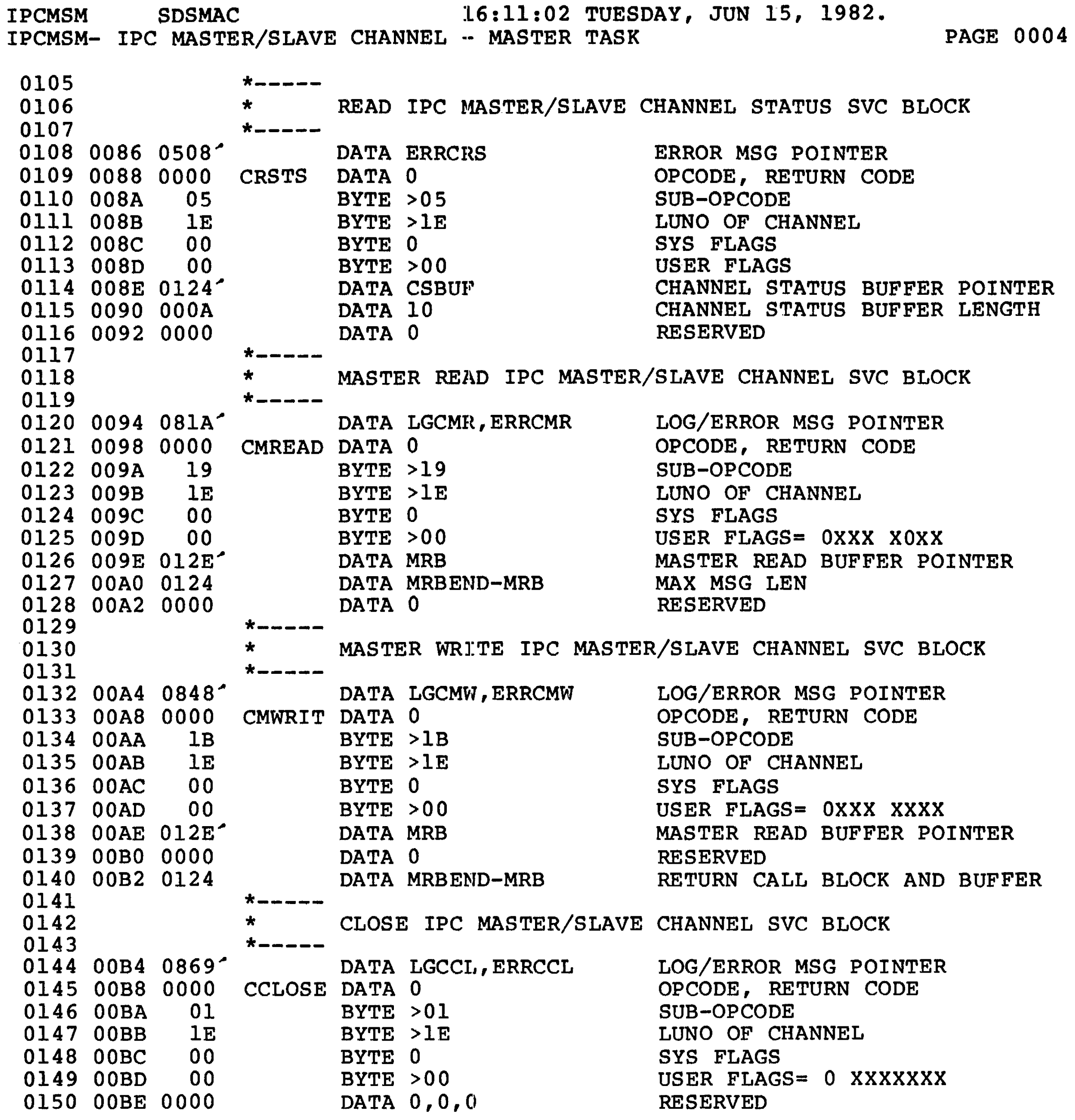
Figure C-1. Master Task (Sheet 1 of 19)

```
IPCMSM     SDSMAC                   16:11:02 TUESDAY, JUN 15, 1982.
IPCMSM- IPC MASTER/SLAVE CHANNEL - MASTER TASK                         PAGE 0002

                  IDT  'IPCMSM'
           *-----
           * NAME:     IPCMSM- IPC MASTER/SLAVE CHANNEL - MASTER TASK
           *
           * ABSTRACT: 1) THIS TASK EXERCISES SOME OF THE OPERATIONS
           *              OF A MASTER/SLAVE IPC CHANNEL FROM THE
           *              VIEWPOINT OF THE MASTER (OWNER) TASK.
           *           2) THIS TASK IS WRITTEN TO BE USED IN
           *              CONJUNCTION WITH THE SLAVE TASK "IPCMSS".
           *           3) THIS TASK CREATES A CHANNEL AS A VDT DEVICE
           *              AND PROCESSES WRITE WITH REPLY REQUESTS
           *              ISSUED BY THE SLAVE TASK "IPCMSS".
           *           4) THIS TASK USES THE VDT ("ME") TO LOG ALL
           *              ACTIVITY ON THE CHANNEL, AND TO ALLOW YOU
           *              TO ENTER THE "REPLY" DATA TO BE RETURNED TO
           *              THE SLAVE TASK.
           * LUNO USE: 1) THIS TASK REQUIRES THE (JOB LOCAL) LUNO >1F
           *              TO BE PRE-ASSIGNED TO THE PROGRAM FILE THAT
           *              CONTAINS THIS TASK.  SEE INSTRUCTIONS BELOW.
           *           2) TASK LOCAL LUNO >1E IS ASSIGNED TO THE
           *              MASTER/SLAVE CHANNEL.
           *           3) TASK LOCAL LUNO >1D IS ASSIGNED TO "ME".
           *
           * ENTRY:    .BID TASK (OR XTS)
           *
           * EXIT:     SVC >04
           *
           * REVISION: 11/13/80 - ORIGINAL
           *
           * ENVIRONMENT: 990 ASSEMBLER
           *
           * EXECUTION REQUIREMENTS:  TO EXECUTE THIS EXAMPLE,
           *           1) ASSIGN LOGICAL NAME (ALN) "VOL1" TO USER DISK
           *              VOLUME NAME.
           *           2) CREATE DIRECTORY (CFDIR) "VOL1.IPC"
           *           3) CREATE PROGRAM FILE (CFPRO) "VOL1.IPC.PROG"
           *           4) ASSEMBLE THIS TASK
           *           5) INSTALL THIS TASK IN "VOL1.IPC.PROG" (TID=01)
           *           6) ASSIGN LUNO (AL) >1F TO "VOL1.IPC.PROG",
           *              SPECIFYING "PROGRAM FILE=YES"
           *           7) EXECUTE THIS TASK VIA XTS
           *           8) TO TERMINATE THE TASK -- WHEN SLAVE SENDS
           *              A STRING BEGINNING WITH "Q", SEND SLAVE A
           *              STRING BEGINNING WITH "Q".
           *-----
0000 0006'        DATA MSMWP,MSM000,0
0006       MSMWP  BSS  32                  WORKSPACE
           *
                  DXOP SVC,15              DEFINE SVC XOP
```

Figure C-1. Master Task (Sheet 2 of 19)

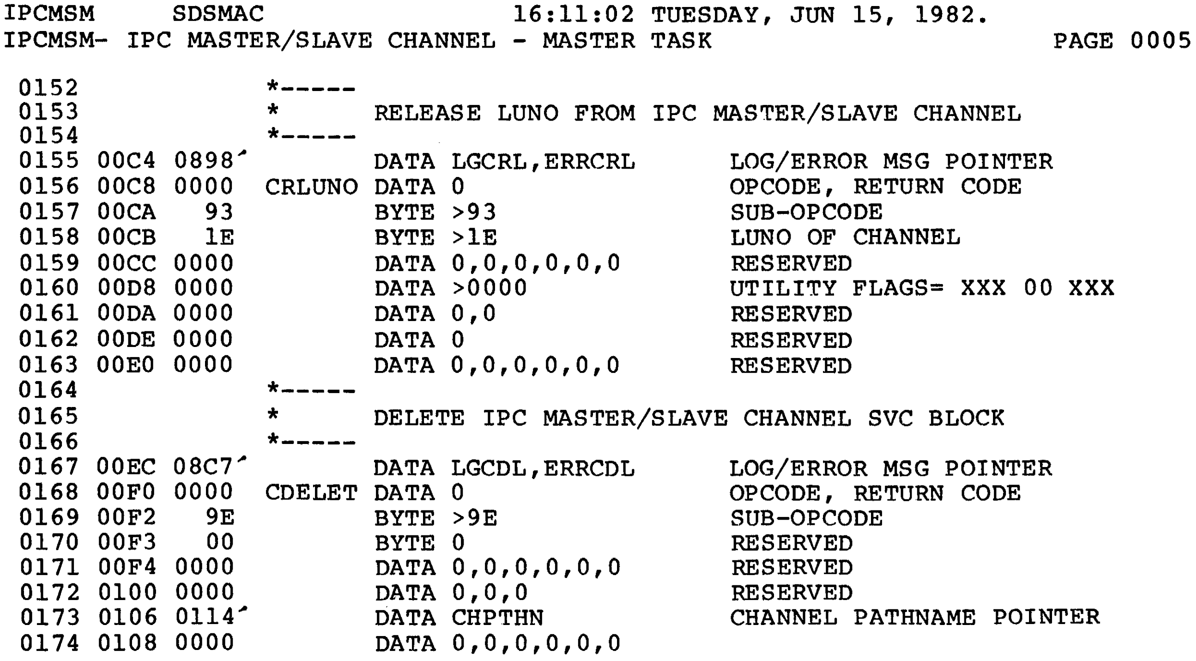
```
IPCMSM     SDSMAC                 16:11:02 TUESDAY, JUN 15, 1982.
IPCMSM- IPC MASTER/SLAVE CHANNEL - MASTER TASK                          PAGE 0003

 0052                *-----
 0053                *     CREATE IPC MASTER/SLAVE CHANNEL SVC BLOCK
 0054                *-----
 0055 0026 075E'           DATA LGCCR,ERRCCR        LOG/ERROR MSG POINTER
 0056 002A 0000 CCREAT DATA 0                   OPCODE, RETURN CODE
 0057 002C   9D            BYTE >9D                 SUB-OPCODE
 0058 002D   1F            BYTE >1F                 PROGRAM FILE LUNO
 0059                *                              (LUNO >1F MUST BE ASSIGNED
 0060                *                              MANUALLY BEFORE EXECUTING TES'
 0061 002E   00            BYTE 0                   SYS FLAGS
 0062 002F   A8            BYTE >A8                 USER FLAGS=1010 1XXX
 0063                *                              BITS 0-1= GLOBAL CHANNEL
 0064                *                              BITS 2   = SHARED CHANNEL
 0065                *                              BITS 3   = MASTER/SLAVE
 0066                *                              BITS 4   = PROCESS ASSIGNS
 0067 0030 0000            DATA 0,0,0,0             RESERVED
 0068 0038   0A            BYTE 10                  DEFLT RESRC TYPE=911
 0069 0039   02            BYTE >02                 DEFLT RESRC FLGS=XXXX X010
 0070                *                              BITS 5-7= DEFAULT RESOURCE
 0071                *                                        IS A DEVICE
 0072 003A 0000            DATA 0                   RESERVED
 0073 003C 0124            DATA MRBEND-MRB          MAX MSG LEN
 0074 003E   01            BYTE >01                 OWNER TASK ID
 0075 003F   00            BYTE 0                   RESERVED
 0076 0040 0114'           DATA CHPTHN              CHANNEL PATHNAME POINTER
 0077 0042 0000            DATA 0,0,0,0,0,0
 0078                *-----
 0079                *     ASSIGN LUNO TO IPC MASTER/SLAVE CHANNEL SVC BLOCK
 0080                *-----
 0081 004E 078D'           DATA LGCAL,ERRCAL        LOG/ERROR MSG POINTER
 0082 0052 0000 CALUNO DATA 0                   OPCODE, RETURN CODE
 0083 0054   91            BYTE >91                 SUB-OPCODE
 0084 0055   1E            BYTE >1E                 LUNO OF CHANNEL
 0085 0056 0000            DATA 0,0,0,0,0,0         RESERVED
 0086 0062 0000            DATA >0000               UTILITY FLAGS=XXX0 0XXX
 0087                *                              BITS 3-4= TASK LOCAL LUNO
 0088 0064 0000            DATA 0,0                 RESERVED
 0089 0068 0114'           DATA CHPTHN              CHANNEL PATHNAME POINTER
 0090 006A 0000            DATA 0,0,0,0,0,0         RESERVED
 0091                *-----
 0092                *     OPEN IPC MASTER/SLAVE CHANNEL SVC BLOCK
 0093                *-----
 0094 0076 07BC'           DATA LGCOP,ERRCOP        LOG/ERROR MSG POINTER
 0095 007A 0000 COPEN  DATA 0                   OPCODE, RETURN CODE
 0096 007C   00            BYTE >00                 SUB-OPCODE
 0097 007D   1E            BYTE >1E                 LUNO OF CHANNEL
 0098 007E   00            BYTE 0                   SYS FLAGS
 0099 007F   10            BYTE >10                 USER FLAGS=0XX1 0XXX
 0100                *                              BIT 3-4= SHARED ACCESS
 0101 0080 0000 COPTYP DATA 0                   RETURNED CHANNEL TYPE
 0102 0082 0000 COPMML DATA 0                   RETURNED MAX MSG LEN
 0103 0084 0000            DATA 0                   RESERVED
```

Figure C-1. Master Task (Sheet 3 of 19)



```
IPCMSM     SDSMAC                    16:11:02 TUESDAY, JUN 15, 1982.
IPCMSM- IPC MASTER/SLAVE CHANNEL -- MASTER TASK                          PAGE 0004

 0105               *-----
 0106               *      READ IPC MASTER/SLAVE CHANNEL STATUS SVC BLOCK
 0107               *-----
 0108 0086 0508'           DATA ERRCRS             ERROR MSG POINTER
 0109 0088 0000   CRSTS    DATA 0                  OPCODE, RETURN CODE
 0110 008A   05            BYTE >05                SUB-OPCODE
 0111 008B   1E            BYTE >1E                LUNO OF CHANNEL
 0112 008C   00            BYTE 0                  SYS FLAGS
 0113 008D   00            BYTE >00                USER FLAGS
 0114 008E 0124'           DATA CSBUF              CHANNEL STATUS BUFFER POINTER
 0115 0090 000A            DATA 10                 CHANNEL STATUS BUFFER LENGTH
 0116 0092 0000            DATA 0                  RESERVED
 0117               *-----
 0118               *      MASTER READ IPC MASTER/SLAVE CHANNEL SVC BLOCK
 0119               *-----
 0120 0094 081A'           DATA LGCMR,ERRCMR       LOG/ERROR MSG POINTER
 0121 0098 0000   CMREAD   DATA 0                  OPCODE, RETURN CODE
 0122 009A   19            BYTE >19                SUB-OPCODE
 0123 009B   1E            BYTE >1E                LUNO OF CHANNEL
 0124 009C   00            BYTE 0                  SYS FLAGS
 0125 009D   00            BYTE >00                USER FLAGS= 0XXX X0XX
 0126 009E 012E'           DATA MRB                MASTER READ BUFFER POINTER
 0127 00A0 0124            DATA MRBEND-MRB         MAX MSG LEN
 0128 00A2 0000            DATA 0                  RESERVED
 0129               *-----
 0130               *      MASTER WRITE IPC MASTER/SLAVE CHANNEL SVC BLOCK
 0131               *-----
 0132 00A4 0848'           DATA LGCMW,ERRCMW       LOG/ERROR MSG POINTER
 0133 00A8 0000   CMWRIT   DATA 0                  OPCODE, RETURN CODE
 0134 00AA   1B            BYTE >1B                SUB-OPCODE
 0135 00AB   1E            BYTE >1E                LUNO OF CHANNEL
 0136 00AC   00            BYTE 0                  SYS FLAGS
 0137 00AD   00            BYTE >00                USER FLAGS= 0XXX XXXX
 0138 00AE 012E'           DATA MRB                MASTER READ BUFFER POINTER
 0139 00B0 0000            DATA 0                  RESERVED
 0140 00B2 0124            DATA MRBEND-MRB         RETURN CALL BLOCK AND BUFFER
 0141               *-----
 0142               *      CLOSE IPC MASTER/SLAVE CHANNEL SVC BLOCK
 0143               *-----
 0144 00B4 0869'           DATA LGCCL,ERRCCL       LOG/ERROR MSG POINTER
 0145 00B8 0000   CCLOSE   DATA 0                  OPCODE, RETURN CODE
 0146 00BA   01            BYTE >01                SUB-OPCODE
 0147 00BB   1E            BYTE >1E                LUNO OF CHANNEL
 0148 00BC   00            BYTE 0                  SYS FLAGS
 0149 00BD   00            BYTE >00                USER FLAGS= 0 XXXXXXX
 0150 00BE 0000            DATA 0,0,0              RESERVED
```

Figure C-1. Master Task (Sheet 4 of 19)



```
IPCMSM      SDSMAC                  16:11:02 TUESDAY, JUN 15, 1982.
IPCMSM- IPC MASTER/SLAVE CHANNEL - MASTER TASK                       PAGE 0005

 0152                *-----
 0153                *      RELEASE LUNO FROM IPC MASTER/SLAVE CHANNEL
 0154                *-----
 0155 00C4 0898'            DATA LGCRL,ERRCRL       LOG/ERROR MSG POINTER
 0156 00C8 0000  CRLUNO DATA 0                      OPCODE, RETURN CODE
 0157 00CA   93             BYTE >93                SUB-OPCODE
 0158 00CB   1E             BYTE >1E                LUNO OF CHANNEL
 0159 00CC 0000             DATA 0,0,0,0,0,0        RESERVED
 0160 00D8 0000             DATA >0000              UTILITY FLAGS= XXX 00 XXX
 0161 00DA 0000             DATA 0,0                RESERVED
 0162 00DE 0000             DATA 0                  RESERVED
 0163 00E0 0000             DATA 0,0,0,0,0,0        RESERVED
 0164                *-----
 0165                *      DELETE IPC MASTER/SLAVE CHANNEL SVC BLOCK
 0166                *-----
 0167 00EC 08C7'            DATA LGCDL,ERRCDL       LOG/ERROR MSG POINTER
 0168 00F0 0000  CDELET DATA 0                      OPCODE, RETURN CODE
 0169 00F2   9E             BYTE >9E                SUB-OPCODE
 0170 00F3   00             BYTE 0                  RESERVED
 0171 00F4 0000             DATA 0,0,0,0,0,0        RESERVED
 0172 0100 0000             DATA 0,0,0              RESERVED
 0173 0106 0114'            DATA CHPTHN             CHANNEL PATHNAME POINTER
 0174 0108 0000             DATA 0,0,0,0,0,0
```

Figure C-1. Master Task (Sheet 5 of 19)

```
IPCMSM     SDSMAC                 16:11:02 TUESDAY, JUN 15, 1982.
IPCMSM- IPC MASTER/SLAVE CHANNEL - MASTER TASK                       PAGE 0006

 0176              *-----
 0177              *      IPC MASTER/SLAVE CHANNEL - PATHNAME
 0178              *-----
 0179 0114    0F   CHPTHN BYTE CNEND-$             CHANNEL PATHNAME LENGTH
 0180 0115    56          TEXT 'VOL1.IPC.MSCHAN'     CHANNEL PATHNAME
 0181        0123' CNEND  EQU  $-1                 CHANNEL PATHNAME END
 0182              *-----
 0183              *      IPC MASTER/SLAVE CHANNEL - READ STATUS BUFFER
 0184              *-----
 0185 0124          CSBUF  EVEN                     CHANNEL STATUS BUFFER
 0186 0124 0000    CSCFLG DATA 0                   CHANNEL FLAGS
 0187 0126    00   CSRTYP BYTE 0                   RESOURCE TYPE
 0188 0127    00   CSRTFL BYTE 0                   RESOURCE TYPE FLAGS
 0189 0128 0000    CSMML  DATA 0                   MAXIMUM MESSAGE LENGTH
 0190 012A    00   CSACT  BYTE 0                   ASSIGN COUNT
 0191 012B    00   CSOCT  BYTE 0                   OPEN COUNT
 0192 012C 0000           DATA 0                   RESERVED
 0193              *-----
 0194              *      IPC MASTER/SLAVE CHANNEL - MASTER READ BUFFER
 0195              *-----
 0196 012E         MRB    EVEN
 0197 012E 0000    MRBSID DATA 0                   SYSTEM SECURITY DATA
 0198 0130 0000    MRBRCB DATA 0                   REQUESTOR CALL BLOCK ADDRESS
 0199 0132 0000    MRBTSB DATA 0                   TSB ADDRESS OF SLAVE TASK
 0200 0134 0000    MRBJSB DATA 0                   JSB ADDRESS OF SLAVE TASK
 0201 0136 0000    MRBSSI DATA 0                   SYSTEM SECURITY INFORMATION
 0202 0138    00   MRBSOC BYTE 0                   I/O OPCODE (>00)
 0203 0139    00   MRBEC  BYTE 0                   I/O RETURN CODE (POSTED BY MAST
 0204 013A    00   MRBOC  BYTE 0                   I/O SUB-OPCODE
 0205 013B    00   MRBLUN BYTE 0                   I/O LUNO
 0206 013C    00   MRBSFL BYTE 0                   SYSTEM FLAGS
 0207         0000 MRFBSY EQU  0                     BUSY
 0208         0001 MRFERR EQU  1                     ERROR
 0209         0002 MRFEOF EQU  2                     END OF FILE
 0210         0003 MRFVNT EQU  3                     EVENT CHAR
 0211 013D    00   MRBUFL BYTE 0                   USER FLAGS
 0212         0000 MRFINT EQU  0                     INITIATE REQUEST
 0213         0001 MRFRPY EQU  1                     WRITE WITH REPLY
 0214         0001 MRFVAL EQU  MRFRPY                READ WITH VALIDATION
 0215              *                                 BITS 2-7 RESERVED
 0216 013E 0000    MRBDBA DATA 0                   DATA BUFFER OFFSET
 0217 0140 0000    MRBICC DATA 0                   READ CHARACTER COUNT
 0218 0142 0000    MRBOCC DATA 0                   WRITE CHAR COUNT/ACTUAL READ CO
 0219 0144 0000    MRBRPY DATA 0                   REPLY BLOCK OFFSET
 0220              *
 0221 0146 0000    MRBRES DATA 0                   RESERVED
 0222 0148 0000    MRBRPA DATA 0                   REPLY BUFFER OFFSET
 0223 014A 0000    MRBRIC DATA 0                   REPLY CHARACTER COUNT
 0224 014C 0000    MRBROC DATA 0                   ACTUAL REPLY COUNT
 0225 014E                BSS  260                 DATA/REPLY BUFFER(S)
 0226        0252' MRBEND EQU  $
```

Figure C-1. Master Task (Sheet 6 of 19)

```
IPCMSM     SDSMAC               16:11:02 TUESDAY, JUN 15, 1982.
IPCMSM- IPC MASTER/SLAVE CHANNEL - MASTER TASK                      PAGE 0007

 0228                *-----
 0229                *      ASSIGN LUNO TO VDT SVC BLOCK
 0230                *-----
 0231 0252 061C'            DATA ERRVAL              ERROR MSG POINTER
 0232 0254 0000      VALUNO DATA 0,>911D             ASSIGN LUNO >1D TO VDT
 0233 0258 0000             DATA 0,0,0,0,0,0         RESERVED
 0234 0264 0000             DATA 0,0,0               RESERVED
 0235 026A 0278'            DATA VPATH               VDT PATHNAME POINTER
 0236 026C 0000             DATA 0,0,0,0,0,0         RESERVED
 0237 0278   02      VPATH  BYTE VPATHE-$            PATHNAME LENGTH
 0238 0279   4D             TEXT 'ME'
 0239      027A'     VPATHE EQU  $-1                 PATHNAME END POINTER
 0240                *-----
 0241                *      OPEN, REWIND VDT SVC BLOCK
 0242                *-----
 0243 027C 064A'            DATA ERRVOP              ERROR MSG POINTER
 0244 027E 0000      VOPEN  DATA 0,>031D             OPEN, REWIND LUNO >1D
 0245 0282 0000             DATA 0,0,0,0             RESERVED
 0246                *-----
 0247                *      WRITE TO VDT SVC BLOCK
 0248                *-----
 0249 028A 0678'            DATA ERRVW               ERROR MSG POINTER
 0250 028C 0000      VWRITE DATA 0,>0B1D             WRITE ASCII TO LUNO >1D
 0251 0290 0000             DATA 0
 0252 0292 0000      VWBUF  DATA $-$                 WRITE BUFFER POINTER
 0253 0294 0000             DATA 0                   RESERVED
 0254 0296 0000      VWLEN  DATA $-$                 WRITE BUFFER LENGTH
 0255                *-----
 0256                *      WRITE (WITH REPLY) VDT SVC BLOCK
 0257                *-----
 0258 0298 06A6'            DATA ERRVWR              ERROR MSG POINTER
 0259 029A 0000      VWRPLY DATA 0,>0B1D             WRITE ASCII TO LUNO >1D
 0260 029E 0040             DATA >0040                 WITH REPLY
 0261 02A0 091E'            DATA LGWRBF              WRITE BUFFER POINTER
 0262 02A2 0000             DATA 0                   RESERVED
 0263 02A4 0019             DATA LGWRBE-LGWRBF       WRITE BUFFER LENGTH
 0264 02A6 02A8'            DATA VWRRBA              REPLY BLOCK ADDRESS
 0265                *
 0266 02A8           VWRRBA EVEN
 0267 02A8 0000      VWRRBF DATA $-$                 REPLY BUFFER POINTER
 0268 02AA 0000      VWRRLN DATA $-$                 REPLY BUFFER LENGTH
 0269 02AC 0000      VWRRCC DATA $-$                 REPLY ACTUAL COUNT
```

Figure C-1. Master Task (Sheet 7 of 19)

```
IPCMSM     SDSMAC                   16:11:02 TUESDAY, JUN 15, 1982.
IPCMSM- IPC MASTER/SLAVE CHANNEL - MASTER TASK                         PAGE 0008

 0271              *-----
 0272              *      CLOSE VDT SVC BLOCK
 0273              *-----
 0274 02AE 06D4'          DATA ERRVCL              ERROR MSG POINTER
 0275 02B0 0000    VCLOSE DATA 0,>011D             CLOSE LUNO >1D
 0276 02B4 0000           DATA 0,0,0,0             RESERVED
 0277              *-----
 0278              *      RELEASE VDT LUNO SVC BLOCK
 0279              *-----
 0280 02BC 0702'          DATA ERRVRL              ERROR MSG POINTER
 0281 02BE 0000    VRLUNO DATA 0,>931D             RELEASE LUNO >1D
 0282 02C2 0000           DATA 0,0,0,0,0,0         RESERVED
 0283 02CE 0000           DATA 0,0,0,0             RESERVED
 0284 02D6 0000           DATA 0,0,0,0,0,0         RESERVED
 0285              *-----
 0286              *      TASK TERMINATION SVC BLOCK
 0287              *-----
 0288 02E2 0400    ETASK  DATA >0400               TASK TERMINATION SVC
 0289              *-----
 0290              *      TIME DELAY SVC BLOCK
 0291              *-----
 0292 02E4 0730'          DATA ERRTD               ERROR MSG POINTER
 0293 02E6 0200    TDELAY DATA >0200               TIME DELAY, RETURN CODE
 0294 02E8 0014           DATA 20                  DELAY COUNT=1 SECOND
 0295              *-----
 0296              *      CONVERT BINARY TO HEX ASCII SVC BLOCK
 0297              *-----
 0298 02EA 0C00    CBHA   DATA >0C00               OPCODE, RETURN CODE
 0299 02EC 0000    CBHA0  DATA $-$                 ASCII DIGIT 0,1
 0300 02EE 0000    CBHA2  DATA $-$                 ASCII DIGIT 2,3
 0301              *-----
 0302              *      MISCELLANEOUS DATA ITEMS
 0303              *-----
 0304 02F0 0000    TDLYCT DATA $-$                 NUMBER OF TIME DELAYS
 0305              *                               BETWEEN "NO ASSIGNS" MSGS
 0306 02F2 0000    ENDTSK DATA 0                   AFTER SLAVE ISSUES 'Q',
 0307              *                               THIS FLAG IS SET TO -1;
 0308              *                               THE NEXT TIME THE SLAVE
 0309              *                               ISSUES A RELEASE LUNO (>93),
 0310              *                               THE MASTER TASK WILL END.
 0311 02F4   51    QUIT   BYTE 'Q'
```

Figure C-1. Master Task (Sheet 8 of 19)

```
IPCMSM     SDSMAC                   16:11:02 TUESDAY, JUN 15, 1982.
IPCMSM- IPC MASTER/SLAVE CHANNEL - MASTER TASK                       PAGE 0009

 0313                 *-----
 0314                 *      TASK ENTRY POINT;
 0315                 *      -ASSIGN, OPEN VDT
 0316                 *      -CREATE, ASSIGN, OPEN MASTER/SLAVE CHANNEL
 0317                 *-----
 0318 02F6            MSM000 EVEN
 0319 02F6 020A              LI    R10,VALUNO
      02F8 0254'
 0320 02FA 06A0              BL    @MSM910          ASSIGN LUNO TO "ME"
      02FC 042C'
 0321 02FE 020A              LI    R10,VOPEN
      0300 027E'
 0322 0302 06A0              BL    @MSM910          OPEN VDT
      0304 042C'
 0323 0306 020A              LI    R10,CCREAT
      0308 002A'
 0324 030A 06A0              BL    @MSM900          CREATE MASTER/SLAVE CHANNEL
      030C 0422'
 0325 030E 020A              LI    R10,CALUNO
      0310 0052'
 0326 0312 06A0              BL    @MSM900          ASSIGN LUNO TO CHANNEL
      0314 0422'
 0327 0316 020A              LI    R10,COPEN
      0318 007A'
 0328 031A 06A0              BL    @MSM900          OPEN CHANNEL
      031C 0422'
 0329 031E 020B              LI    R11,10           SET TO LOG "NO ASSIGNS"
      0320 000A
 0330 0322 C80B              MOV   R11,@TDLYCT        MSG EVERY 10 SECONDS
      0324 02F0'
```

Figure C-1. Master Task (Sheet 9 of 19)

```
IPCMSM     SDSMAC                 16:11:02 TUESDAY, JUN 15, 1982.
IPCMSM- IPC MASTER/SLAVE CHANNEL - MASTER TASK                        PAGE 0010

 0332              *-----
 0333              *      READ CHANNEL STATUS; IF ASSIGN COUNT = 1,
 0334              *      GO DO OTHER PROCESSING, THEN COME BACK.
 0335              *      (NOTE: THE ASSIGN AND OPEN COUNTS WILL ALWAYS
 0336              *      BE AT LEAST 1, TO ACCOUNT FOR THE MASTER TASK)
 0337              *-----
 0338 0326 020A  MSM100 LI    R10,CRSTS
      0328 0088'
 0339 032A 06A0         BL    @MSM910          READ CHANNEL STATUS
      032C 042C'
 0340 032E D020         MOVB  @CSACT,R0        GET ASSIGN COUNT
      0330 012A'
 0341 0332 0990         SRL   R0,9             COUNT > 1?
 0342 0334 1510         JGT   MSM200           YES, GO DO MASTER READ
 0343              *-----
 0344              *      TIME DELAY LOOP; DELAY 1 SECOND, THEN RE-ISSUE
 0345              *      READ CHANNEL STATUS; LOG "NO ASSIGNS" MSG TO VDT
 0346              *      EVERY 10 SECONDS
 0347              *-----
 0348 0336 0620         DEC   @TDLYCT          TIME TO LOG AGAIN?
      0338 02F0'
 0349 033A 1508         JGT   MSM150           NO
 0350 033C 020B         LI    R11,10           YES, RESET COUNT
      033E 000A
 0351 0340 C80B         MOV   R11,@TDLYCT        TO 10
      0342 02F0'
 0352 0344 0209         LI    R9,LGNOA
      0346 07EB'
 0353 0348 06A0         BL    @MSM920          LOG TIME DELAY MSG
      034A 043A'
 0354 034C 020A  MSM150 LI    R10,TDELAY
      034E 02E6'
 0355 0350 06A0         BL    @MSM910          TIME DELAY FOR 1 SECOND
      0352 042C'
 0356 0354 10E8         JMP   MSM100           TIME FOR A CHANNEL CHECK AGAIN
```

Figure C-1. Master Task (Sheet 10 of 19)

```
IPCMSM     SDSMAC                  16:11:02 TUESDAY, JUN 15, 1982.
IPCMSM- IPC MASTER/SLAVE CHANNEL - MASTER TASK                    PAGE 0011

 0358                *-----
 0359                *      A SLAVE TASK HAS ISSUED AN ASSIGN;
 0360                *      DO MASTER READ AND PROCESS THE SLAVE I/O REQ.
 0361                *----
 0362 0356 020A  MSM200 LI    R10,CMREAD
      0358 0098'
 0363 035A 06A0         BL    @MSM910          MASTER READ CHANNEL
      035C 042C'
 0364 035E D020         MOVB  @MRBOC,R0        GET SLAVE I/O SUB-OPCODE
      0360 013A'
 0365 0362 0980         SRL   R0,8             SHIFT TO LSB
 0366 0364 C0C0         MOV   R0,R3            SAVE FOR LATER COMPARE
 0367 0366 2FE0         SVC   @CBHA            CONVERT OPCODE TO HEX
      0368 02EA'
 0368 036A C820         MOV   @CBHA2,@LGCMRO     AND MOVE TO M/R MSG
      036C 02EE'
      036E 0846'
 0369 0370 0209         LI    R9,LGCMR         POINT TO LOG MSG
      0372 081A'
 0370 0374 06A0         BL    @MSM920          LOG MASTER READ MSG
      0376 043A'
 0371 0378 0283         CI    R3,>0093         RELEASE LUNO?
      037A 0093
 0372 037C 1603         JNE   MSM210           NO
 0373 037E C020         MOV   @ENDTSK,R0       END MASTER TASK FLAG SET?
      0380 02F2'
 0374 0382 1135         JLT   MSM700           YES, END MASTER TASK
 0375 0384 0283  MSM210 CI    R3,>000B         SLAVE REQUESTING WRITE?
      0386 000B
 0376 0388 1305         JEQ   MSM250           YES, GO CHECK FOR REPLY
 0377 038A 020A  MSM220 LI    R10,CMWRIT
      038C 00A8'
 0378 038E 06A0         BL    @MSM900          MASTER WRITE CHANNEL
      0390 0422'
 0379 0392 10C9         JMP   MSM100           GO CHECK FOR MORE SLAVES
```

Figure C-1. Master Task (Sheet 11 of 19)

```
 IPCMSM     SDSMAC                   16:11:02 TUESDAY, JUN 15, 1982.
 IPCMSM- IPC MASTER/SLAVE CHANNEL - MASTER TASK                       PAGE 0012

  0381                 *-----
  0382                 *      SLAVE HAS REQUESTED A WRITE ASCII;
  0383                 *      LOG THE WRITE BUFFER, AND CHECK FOR REPLY.
  0384                 *      IF REPLY, ACCEPT RESPONSE FROM MASTER VDT,
  0385                 *      THEN SATISIFY THE REPLY.
  0386                 *-----
  0387 0394 0209  MSM250 LI    R9,LGWD             POINT TO LOG MSG
       0396 08F6'
  0388 0398 06A0         BL    @MSM920             LOG "SLAVE WRITE" MSG
       039A 043A'
  0389 039C 0209         LI    R9,MRB              POINT TO
       039E 012E'
  0390 03A0 A260         A     @MRBDBA,R9            WRITE BUFFER IN MRB
       03A2 013E'
  0391 03A4 9819         CB    *R9,@QUIT           REQUEST TO END MASTER TASK?
       03A6 02F4'
  0392 03A8 1602         JNE   MSM260              NO
  0393 03AA 0720         SETO  @ENDTSK             YES, SET END TASK FLAG
       03AC 02F2'
  0394 03AE C809  MSM260 MOV   R9,@VWBUF           SET WRITE BUFFER POINTER
       03B0 0292'
  0395 03B2 C820         MOV   @MRBOCC,@VWLEN      SET WRITE LENGTH
       03B4 0142'
       03B6 0296'
  0396 03B8 2FE0         SVC   @VWRITE             LOG SLAVE WRITE BUFFER
       03BA 028C'
  0397 03BC D020         MOVB  @MRBUFL,R0          GET USER FLAGS
       03BE 013D'
  0398 03C0 0A20         SLA   R0,MRFRPY+1         SHIFT OUT REPLY FLAG (BIT 2)
  0399                 *                           WAS REPLY FLAG ON?
  0400 03C2 17E3         JNC   MSM220              NO, GO DO MASTER WRITE
  0401 03C4 C820         MOV   @MRBDBA,@MRBRPA     ESTABLISH REPLY BUFFER OFFSET
       03C6 013E'
       03C8 0148'
  0402 03CA 0208         LI    R8,MRB              YES, POINT TO
       03CC 012E'
  0403 03CE A220         A     @MRBRPY,R8            REPLY BLOCK IN MRB
       03D0 0144'
  0404 03D2 C038         MOV   *R8+,R0             GET REPLY BUFFER OFFSET
  0405 03D4 0220         AI    R0,MRB              MAKE POINTER ABSOLUTE
       03D6 012E'
  0406 03D8 C800         MOV   R0,@VWRRBF          SET REPLY BUFFER POINTER
       03DA 02A8'
  0407 03DC C838         MOV   *R8+,@VWRRLN        SET REPLY BUFFER LENGTH
       03DE 02AA'
  0408 03E0 020A         LI    R10,VWRPLY
       03E2 029A'
  0409 03E4 06A0         BL    @MSM910             WRITE VDT WITH REPLY
       03E6 042C'
  0410 03E8 C620         MOV   @VWRRCC,*R8         SET ACTUAL READ COUNT IN MRB
       03EA 02AC'
  0411 03EC 10CE         JMP   MSM220              GO DO MASTER WRITE
```

Figure C-1. Master Task (Sheet 12 of 19)

```
IPCMSM     SDSMAC                 16:11:02 TUESDAY, JUN 15, 1982.
IPCMSM- IPC MASTER/SLAVE CHANNEL - MASTER TASK                       PAGE 0013

 0413               *----
 0414               *      END MASTER TASK
 0415               *-----
 0416 03EE 020A    MSM700 LI   R10,CMWRIT
      03F0 00A8'
 0417 03F2 06A0           BL   @MSM900         MASTER WRITE CHANNEL
      03F4 0422'
 0418 03F6 020A           LI   R10,CCLOSE
      03F8 00B8'
 0419 03FA 06A0           BL   @MSM900         CLOSE MASTER/SLAVE CHANNEL
      03FC 0422'
 0420 03FE 020A           LI   R10,CRLUNO
      0400 00C8'
 0421 0402 06A0           BL   @MSM900         RELEASE LUNO TO CHANNEL
      0404 0422'
 0422 0406 020A           LI   R10,CDELET
      0408 00F0'
 0423 040A 06A0           BL   @MSM900         DELETE MASTER/SLAVE CHANNEL
      040C 0422'
 0424 040E 020A           LI   R10,VCLOSE
      0410 02B0'
 0425 0412 06A0           BL   @MSM910         CLOSE VDT
      0414 042C'
 0426 0416 020A           LI   R10,VRLUNO
      0418 02BE'
 0427 041A 06A0           BL   @MSM910         RELEASE VDT LUNO
      041C 042C'
 0428 041E 2FE0           SVC  @ETASK          TERMINATE TASK
      0420 02E2'
```

Figure C-1. Master Task (Sheet 13 of 19)

```
IPCMSM     SDSMAC                  16:11:02 TUESDAY, JUN 15, 1982.
IPCMSM- IPC MASTER/SLAVE CHANNEL - MASTER TASK                     PAGE 0014

 0430                *-----
 0431                *      ISSUE SVC AND CHECK FOR ERROR UPON COMPLETION
 0432                *      ENTRY POINTS: MSM900 DO I/O AND LOG TO VDT
 0433                *                    MSM910 DO I/O ONLY (NO LOGGING)
 0434                *                    MSM920 LOG ONLY (NO I/O)
 0435                *-----
 0436 0422 C24A  MSM900 MOV  R10,R9              POINT TO
 0437 0424 0229         AI   R9,-4                 MESSAGE TO BE LOGGED
      0426 FFFC
 0438 0428 C259         MOV  *R9,R9                  UPON SUCCESSFUL COMPLETION
 0439 042A 1001         JMP  MSM915
 0440 042C 04C9  MSM910 CLR  R9                  SET DUMMY LOG POINTER
 0441 042E 2FDA  MSM915 SVC  *R10                ISSUE SVC
 0442 0430 D02A         MOVB @1(R10),R0          GET RETURN CODE; ANY ERROR?
      0432 0001
 0443 0434 160B         JNE  MSM960              YES, GO SEE IF ERR MSG NEEDED
 0444 0436 C249         MOV  R9,R9               MSG TO BE LOGGED?
 0445 0438 1308         JEQ  MSM950              NO
 0446                *-----
 0447                *      LOG THE I/O (JUST COMPLETED) ON THE VDT
 0448                *-----
 0449 043A D039  MSM920 MOVB *R9+,R0             GET MSG LENGTH
 0450 043C 0980         SRL  R0,8                SHIFT TO LSB
 0451 043E C800         MOV  R0,@VWLEN           SET WRITE LENGTH
      0440 0296'
 0452 0442 C809         MOV  R9,@VWBUF           SET BUFFER POINTER
      0444 0292'
 0453 0446 2FE0         SVC  @VWRITE             LOG MSG TO VDT
      0448 028C'
 0454 044A 045B  MSM950 RT                       NO, EXIT
 0455 044C C06A  MSM960 MOV  @-2(R10),R1         IS ERROR MSG NEEDED?
      044E FFFE
 0456 0450 13FC         JEQ  MSM950              NO, EXIT
 0457 0452 0980         SRL  R0,8                SHIFT ERROR CODE TO LSB
 0458 0454 2FE0         SVC  @CBHA               CONVERT TO HEX ASCII
      0456 02EA'
 0459 0458 D020         MOVB @CBHA+1,R0          CONVERT OK?
      045A 02EB'
 0460 045C 1606         JNE  MSM990              NO, DON'T PLUG INTO MSG
 0461 045E C860         MOV  @CBHA0,@10(R1)      MOVE HEX ERR
      0460 02EC'
      0462 000A
 0462 0464 C860         MOV  @CBHA2,@12(R1)        TO ERROR MSG
      0466 02EE'
      0468 000C
 0463 046A C801  MSM990 MOV  R1,@VWBUF           SET BUFFER POINTER
      046C 0292'
 0464 046E 0201         LI   R1,ERRLEN           SET ERROR
      0470 002D
 0465 0472 C801         MOV  R1,@VWLEN             MESSAGE LENGTH
      0474 0296'
 0466 0476 2FE0         SVC  @VWRITE             LOG ERROR MSG ON VDT
      0478 028C'
 0467 047A 2FE0         SVC  @ETASK              TERMINATE TASK
      047C 02E2'
```

Figure C-1. Master Task (Sheet 14 of 19)

```
IPCMSM     SDSMAC                    16:11:02 TUESDAY, JUN 15, 1982.
IPCMSM- IPC MASTER/SLAVE CHANNEL - MASTER TASK                     PAGE 0015

 0469                *-----
 0470                *      I/O ERROR MESSAGES
 0471                *-----
 0472 047E 0D0A   ERRCCR DATA >0D0A
 0473 0480   49          TEXT 'IPCMSM- XXXX - ERROR ON CREATE CHANNEL         '
 0474      002D   ERRLEN EQU  $-ERRCCR-1
 0475 04AC 0D0A   ERRCAL DATA >0D0A
 0476 04AE   49          TEXT 'IPCMSM- XXXX - ERROR ON ASSIGN CHANNEL LUNO '
 0477 04DA 0D0A   ERRCOP DATA >0D0A
 0478 04DC   49          TEXT 'IPCMSM- XXXX - ERROR ON OPEN CHANNEL           '
 0479 0508 0D0A   ERRCRS DATA >0D0A
 0480 050A   49          TEXT 'IPCMSM- XXXX - ERROR ON READ CHANNEL STATUS '
 0481 0536 0D0A   ERRCMR DATA >0D0A
 0482 0538   49          TEXT 'IPCMSM- XXXX - ERROR ON MASTER READ CHANNEL '
 0483 0564 0D0A   ERRCMW DATA >0D0A
 0484 0566   49          TEXT 'IPCMSM- XXXX - ERROR ON MASTER WRITE CHANNEL'
 0485 0592 0D0A   ERRCCL DATA >0D0A
 0486 0594   49          TEXT 'IPCMSM- XXXX - ERROR ON CLOSE CHANNEL          '
 0487 05C0 0D0A   ERRCRL DATA >0D0A
 0488 05C2   49          TEXT 'IPCMSM- XXXX - ERROR ON RELEASE CHANNEL LUNO'
 0489 05EE 0D0A   ERRCDL DATA >0D0A
 0490 05F0   49          TEXT 'IPCMSM- XXXX - ERROR ON DELETE CHANNEL         '
 0491                *
 0492 061C 0D0A   ERRVAL DATA >0D0A
 0493 061E   49          TEXT 'IPCMSM- XXXX - ERROR ON ASSIGN VDT LUNO        '
 0494 064A 0D0A   ERRVOP DATA >0D0A
 0495 064C   49          TEXT 'IPCMSM- XXXX - ERROR ON OPEN VDT               '
 0496 0678 0D0A   ERRVW  DATA >0D0A
 0497 067A   49          TEXT 'IPCMSM- XXXX - ERROR ON WRITE VDT              '
 0498 06A6 0D0A   ERRVWR DATA >0D0A
 0499 06A8   49          TEXT 'IPCMSM- XXXX - ERROR ON WRITE VDT W/REPLY      '
 0500 06D4 0D0A   ERRVCL DATA >0D0A
 0501 06D6   49          TEXT 'IPCMSM- XXXX - ERROR ON CLOSE VDT              '
 0502 0702 0D0A   ERRVRL DATA >0D0A
 0503 0704   49          TEXT 'IPCMSM- XXXX - ERROR ON RELEASE VDT LUNO       '
 0504                *
 0505 0730 0D0A   ERRTD  DATA >0D0A
 0506 0732   49          TEXT 'IPCMSM- XXXX - ERROR ON TIME DELAY             '
```

Figure C-1. Master Task (Sheet 15 of 19)

```
IPCMSM     SDSMAC                  16:11:02 TUESDAY, JUN 15, 1982.
IPCMSM- IPC MASTER/SLAVE CHANNEL -- MASTER TASK                    PAGE 0016

 0508                *-----
 0509                *      CHANNEL I/O ACTIVITY MESSAGES LOGGED TO VDT
 0510                *-----
 0511 075E    2E     LGCCR  BYTE LGCCRE-$
 0512 075F    0D            BYTE >0D,>0A
 0513 0761    49            TEXT 'IPCMSM-001 IPC MASTER/SLAVE CHANNEL CREATED '
 0514       078C'  LGCCRE EQU  $-1
 0515                *
 0516 078D    2E     LGCAL  BYTE LGCALE-$
 0517 078E    0D            BYTE >0D,>0A
 0518 0790    49            TEXT 'IPCMSM-002 IPC MASTER/SLAVE CHANNEL ASSIGNED'
 0519       07BB'  LGCALE EQU  $-1
 0520                *
 0521 07BC    2E     LGCOP  BYTE LGCOPE-$
 0522 07BD    0D            BYTE >0D,>0A
 0523 07BF    49            TEXT 'IPCMSM-003 IPC MASTER/SLAVE CHANNEL OPENED  '
 0524       07EA'  LGCOPE EQU  $-1
 0525                *
 0526 07EB    2E     LGNOA  BYTE LGNOAE-$
 0527 07EC    0D            BYTE >0D,>0A
 0528 07EE    49            TEXT 'IPCMSM-004 NO ASSIGNS ON IPC CHANNEL        '
 0529       0819'  LGNOAE EQU  $-1
 0530                *
 0531 081A    2D     LGCMR  BYTE LGCMRE-$
 0532 081B    0D            BYTE >0D,>0A
 0533 081D    49            TEXT 'IPCMSM-005 MASTER READ; I/O SUB-OPCODE= '
 0534 0846 0000     LGCMRO DATA $-$
 0535       0847'  LGCMRE EQU  $-1
 0536                *
 0537 0848    20     LGCMW  BYTE LGCMWE-$
 0538 0849    0D            BYTE >0D,>0A
 0539 084B    49            TEXT 'IPCMSM-006 MASTER WRITE ISSUED'
 0540       0868'  LGCMWE EQU  $-1
 0541                *
 0542 0869    2E     LGCCL  BYTE LGCCLE-$
 0543 086A    0D            BYTE >0D,>0A
 0544 086C    49            TEXT 'IPCMSM-007 IPC MASTER/SLAVE CHANNEL CLOSED  '
 0545       0897'  LGCCLE EQU  $-1
 0546                *
 0547 0898    2E     LGCRL  BYTE LGCRLE-$
 0548 0899    0D            BYTE >0D,>0A
 0549 089B    49            TEXT 'IPCMSM-008 IPC MASTER/SLAVE CHANNEL RELEASED'
 0550       08C6'  LGCRLE EQU  $-1
 0551                *
 0552 08C7    2E     LGCDL  BYTE LGCDLE-$
 0553 08C8    0D            BYTE >0D,>0A
 0554 08CA    49            TEXT 'IPCMSM-009 IPC MASTER/SLAVE CHANNEL DELETED '
 0555       08F5'  LGCDLE EQU  $-1
 0556                *
 0557 08F6    26     LGWD   BYTE LGWDE-$
 0558 08F7    0D            BYTE >0D,>0A
 0559 08F9    49            TEXT 'IPCMSM-010 DATA WRITTEN FROM SLAVE: '
 0560       091C'  LGWDE  EQU  $-1
 0561                *
 0562 091E 0A0D     LGWRBF DATA >0A0D
 0563 0920    49            TEXT 'IPCMSM-011 ENTER REPLY: '
 0564       0937'  LGWRBE EQU  $-1
 0565                       END
NO ERRORS,      NO WARNINGS
```

Figure C-1. Master Task (Sheet 16 of 19)

```
IPCMSM    SDSMAC                  16:11:02 TUESDAY, JUN 15, 1982.
LABEL     VALUE   DEFN  REFERENCES                                          PAGE 0017

$          0938'        0179  0181  0226  0237  0239  0252  0252  0254  0254
                        0267  0267  0268  0268  0269  0269  0299  0299  0300
                        0300  0304  0304  0474  0511  0514  0516  0519  0521
                        0524  0526  0529  0531  0534  0534  0535  0537  0540
                        0542  0545  0547  0550  0552  0555  0557  0560  0564
CALUNO     0052'  0082  0325
CBHA       02EA'  0298  0367  0458  0459
CBHA0      02EC'  0299  0461
CBHA2      02EE'  0300  0368  0462
CCLOSE     00B8'  0145  0418
CCREAT     002A'  0056  0323
CDELET     00F0'  0168  0422
CHPTHN     0114'  0179  0076  0089  0173
CMREAD     0098'  0121  0362
CMWRIT     00A8'  0133  0377  0416
CNEND      0123'  0181  0179
COPEN      007A'  0095  0327
COPMML     0082'  0102
COPTYP     0080'  0101
CRLUNO     00C8'  0156  0420
CRSTS      0088'  0109  0338
CSACT      012A'  0190  0340
CSBUF      0124'  0185  0114
CSCFLG     0124'  0186
CSMML      0128'  0189
CSOCT      012B'  0191
CSRTFL     0127'  0188
CSRTYP     0126'  0187
ENDTSK     02F2'  0306  0373  0393
ERRCAL     04AC'  0475  0081
ERRCCL     0592'  0485  0144
ERRCCR     047E'  0472  0055  0474
ERRCDL     05EE'  0489  0167
ERRCMR     0536'  0481  0120
ERRCMW     0564'  0483  0132
ERRCOP     04DA'  0477  0094
ERRCRL     05C0'  0487  0155
ERRCRS     0508'  0479  0108
ERRLEN     002D   0474  0464
ERRTD      0730'  0505  0292
ERRVAL     061C'  0492  0231
ERRVCL     06D4'  0500  0274
ERRVOP     064A'  0494  0243
ERRVRL     0702'  0502  0280
ERRVW      0678'  0496  0249
ERRVWR     06A6'  0498  0258
ETASK      02E2'  0288  0428  0467
LGCAL      078D'  0516  0081
LGCALE     07BB'  0519  0516
LGCCL      0869'  0542  0144
LGCCLE     0897'  0545  0542
LGCCR      075E'  0511  0055
LGCCRE     078C'  0514  0511
LGCDL      08C7'  0552  0167
LGCDLE     08F5'  0555  0552
LGCMR      081A'  0531  0120  0369
LGCMRE     0847'  0535  0531
LGCMRO     0846'  0534  0368
LGCMW      0848'  0537  0132
LGCMWE     0868'  0540  0537
```

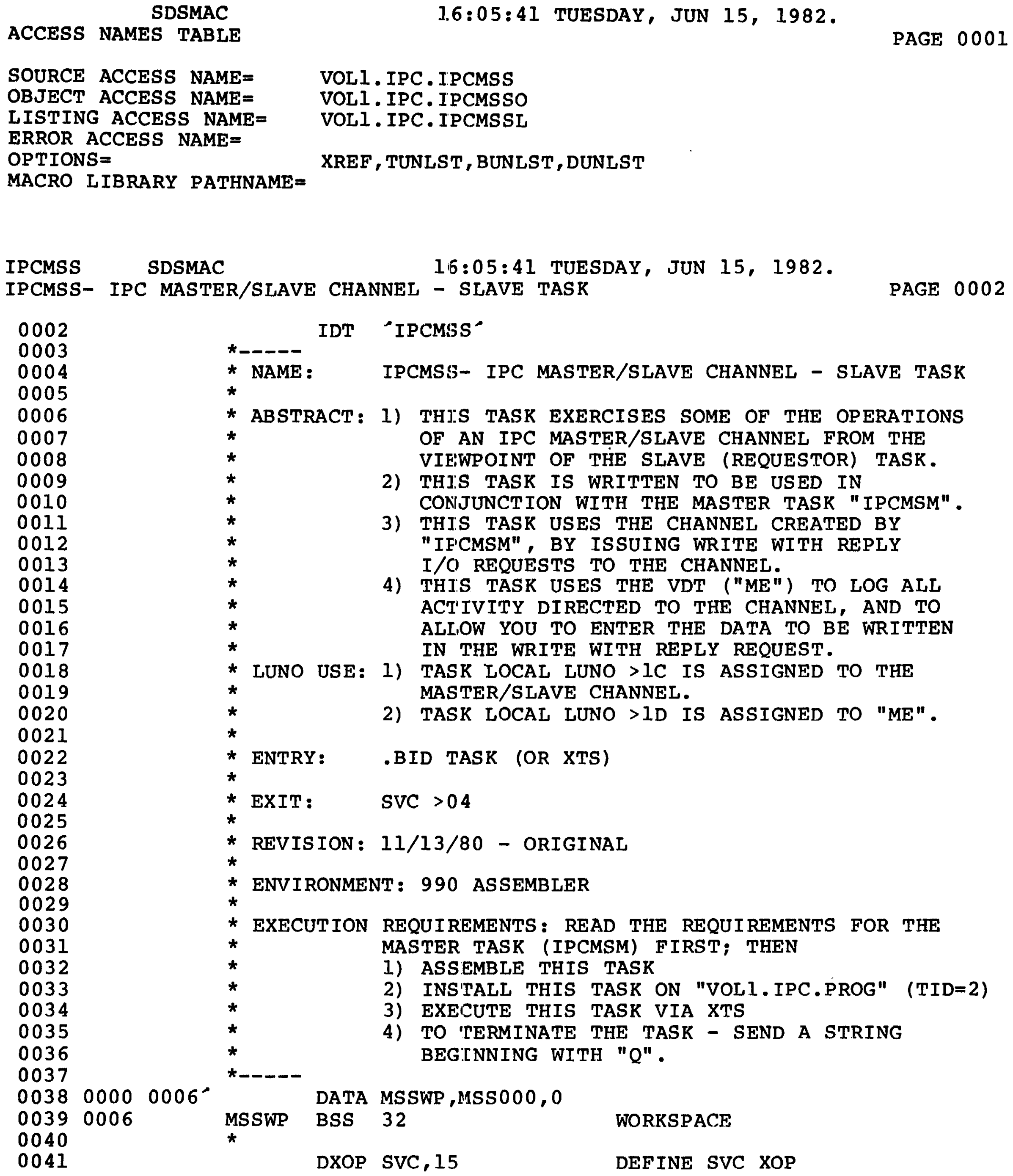
Figure C-1.   Master Task (Sheet 17 of 19)

```
IPCMSM     SDSMAC 3.4.0 81.117   16:11:02 TUESDAY, JUN 15, 1982.
LABEL      VALUE    DEFN  REFERENCES                                          PAGE 0018

LGCOP      07BC'    0521  0094
LGCOPE     07EA'    0524  0521
LGCRL      0898'    0547  0155
LGCRLE     08C6'    0550  0547
LGNOA      07EB'    0526  0352
LGNOAE     0819'    0529  0526
LGWD       08F6'    0557  0387
LGWDE      091C'    0560  0557
LGWRBE     0937'    0564  0263
LGWRBF     091E'    0562  0261  0263
MRB        012E'    0196  0073  0126  0127  0138  0140  0389  0402  0405
MRBDBA     013E'    0216  0390  0401
MRBEC      0139'    0203
MRBEND     0252'    0226  0073  0127  0140
MRBICC     0140'    0217
MRBJSB     0134'    0200
MRBLUN     013B'    0205
MRBOC      013A'    0204  0364
MRBOCC     0142'    0218  0395
MRBRCB     0130'    0198
MRBRES     0146'    0221
MRBRIC     014A'    0223
MRBROC     014C'    0224
MRBRPA     0148'    0222  0401
MRBRPY     0144'    0219  0403
MRBSFL     013C'    0206
MRBSID     012E'    0197
MRBSOC     0138'    0202
MRBSSI     0136'    0201
MRBTSB     0132'    0199
MRBUFL     013D'    0211  0397
MRFBSY     0000     0207
MRFEOF     0002     0209
MRFERR     0001     0208
MRFINT     0000     0212
MRFRPY     0001     0213  0214  0398
MRFVAL     0001     0214
MRFVNT     0003     0210
MSM000     02F6'    0318  0047
MSM100     0326'    0338  0356  0379
MSM150     034C'    0354  0349
MSM200     0356'    0362  0342
MSM210     0384'    0375  0372
MSM220     038A'    0377  0400  0411
MSM250     0394'    0387  0376
MSM260     03AE'    0394  0392
MSM700     03EE'    0416  0374
MSM900     0422'    0436  0324  0326  0328  0378  0417  0419  0421  0423
MSM910     042C'    0440  0320  0322  0339  0355  0363  0409  0425  0427
MSM915     042E'    0441  0439
MSM920     043A'    0449  0353  0370  0388
MSM950     044A'    0454  0445  0456
MSM960     044C'    0455  0443
MSM990     046A'    0463  0460
MSMWP      0006'    0048  0047
QUIT       02F4'    0311  0391
R0         0000           0340  0341  0364  0365  0366  0373  0397  0398  0404
                          0405  0406  0442  0449  0450  0451  0457  0459
R1         0001           0455  0461  0462  0463  0464  0465
R10        000A           0319  0321  0323  0325  0327  0338  0354  0362  0377
```

Figure C-1.' Master Task (Sheet 18 of 19)

```
IPCMSM      SDSMAC                      16:11:02 TUESDAY, JUN 15, 1982.
LABEL      VALUE   DEFN  REFERENCES                                              PAGE 0019

                         0408   0416   0418   0420   0422   0424   0426   0436   0441
                         0442   0455
 R11        000B         0329   0330   0350   0351
 R3         0003         0366   0371   0375
 R8         0008         0402   0403   0404   0407   0410
 R9         0009         0352   0369   0387   0389   0390   0391   0394   0436   0437
                         0438   0438   0440   0444   0444   0449   0452
 SVC               0050
 TDELAY     02E6'  0293  0354
 TDLYCT     02F0'  0304  0330   0348   0351
 VALUNO     0254'  0232  0319
 VCLOSE     02B0'  0275  0424
 VOPEN      027E'  0244  0321
 VPATH      0278'  0237  0235
 VPATHE     027A'  0239  0237
 VRLUNO     02BE'  0281  0426
 VWBUF      0292'  0252  0394   0452   0463
 VWLEN      0296'  0254  0395   0451   0465
 VWRITE     028C'  0250  0396   0453   0466
 VWRPLY     029A'  0259  0408
 VWRRBA     02A8'  0266  0264
 VWRRBF     02A8'  0267  0406
 VWRRCC     02AC'  0269  0410
 VWRRLN     02AA'  0268  0407
```

Figure C-1. Master Task (Sheet 19 of 19)



```
          SDSMAC                    16:05:41 TUESDAY, JUN 15, 1982.
 ACCESS NAMES TABLE                                                    PAGE 0001

 SOURCE ACCESS NAME=      VOL1.IPC.IPCMSS
 OBJECT ACCESS NAME=      VOL1.IPC.IPCMSSO
 LISTING ACCESS NAME=     VOL1.IPC.IPCMSSL
 ERROR ACCESS NAME=
 OPTIONS=                 XREF,TUNLST,BUNLST,DUNLST
 MACRO LIBRARY PATHNAME=


IPCMSS     SDSMAC                   16:05:41 TUESDAY, JUN 15, 1982.
IPCMSS- IPC MASTER/SLAVE CHANNEL - SLAVE TASK                          PAGE 0002

                        IDT  'IPCMSS'
                 *-----
                 * NAME:     IPCMSS- IPC MASTER/SLAVE CHANNEL - SLAVE TASK
                 *
                 * ABSTRACT: 1) THIS TASK EXERCISES SOME OF THE OPERATIONS
                 *              OF AN IPC MASTER/SLAVE CHANNEL FROM THE
                 *              VIEWPOINT OF THE SLAVE (REQUESTOR) TASK.
                 *           2) THIS TASK IS WRITTEN TO BE USED IN
                 *              CONJUNCTION WITH THE MASTER TASK "IPCMSM".
                 *           3) THIS TASK USES THE CHANNEL CREATED BY
                 *              "IPCMSM", BY ISSUING WRITE WITH REPLY
                 *              I/O REQUESTS TO THE CHANNEL.
                 *           4) THIS TASK USES THE VDT ("ME") TO LOG ALL
                 *              ACTIVITY DIRECTED TO THE CHANNEL, AND TO
                 *              ALLOW YOU TO ENTER THE DATA TO BE WRITTEN
                 *              IN THE WRITE WITH REPLY REQUEST.
                 * LUNO USE: 1) TASK LOCAL LUNO >1C IS ASSIGNED TO THE
                 *              MASTER/SLAVE CHANNEL.
                 *           2) TASK LOCAL LUNO >1D IS ASSIGNED TO "ME".
                 *
                 * ENTRY:    .BID TASK (OR XTS)
                 *
                 * EXIT:     SVC >04
                 *
                 * REVISION: 11/13/80 - ORIGINAL
                 *
                 * ENVIRONMENT: 990 ASSEMBLER
                 *
                 * EXECUTION REQUIREMENTS: READ THE REQUIREMENTS FOR THE
                 *           MASTER TASK (IPCMSM) FIRST; THEN
                 *           1) ASSEMBLE THIS TASK
                 *           2) INSTALL THIS TASK ON "VOL1.IPC.PROG" (TID=2)
                 *           3) EXECUTE THIS TASK VIA XTS
                 *           4) TO TERMINATE THE TASK - SEND A STRING
                 *              BEGINNING WITH "Q".
                 *-----
     0000 0006'         DATA MSSWP,MSS000,0
     0006        MSSWP  BSS  32               WORKSPACE
                 *
                        DXOP SVC,15           DEFINE SVC XOP
```

Figure C-2. Slave Task (Sheet 1 of 12)

```
IPCMSS     SDSMAC                 16:05:41 TUESDAY, JUN 15, 1982.
IPCMSS- IPC MASTER/SLAVE CHANNEL - SLAVE TASK                          PAGE 0003

            *-----
            *      ASSIGN LUNO TO IPC MASTER/SLAVE CHANNEL SVC BLOCK
            *-----
0026 048C'         DATA LGCAL,ERRCAL      LOG/ERROR MSG POINTER
002A 0000   CALUNO DATA 0                 OPCODE, RETURN CODE
002C   91          BYTE >91               SUB-OPCODE
002D   1C          BYTE >1C               LUNO OF CHANNEL
002E 0000          DATA 0,0,0,0,0,0       RESERVED
003A 0000          DATA >0000             UTILITY FLAGS=XXX0 0XXX
            *                             BITS 3-4= TASK LOCAL LUNO
003C 0000          DATA 0,0               RESERVED
0040 00AE'         DATA CHPTHN            CHANNEL PATHNAME POINTER
0042 0000          DATA 0,0,0,0,0,0       RESERVED
            *-----
            *      OPEN IPC MASTER/SLAVE CHANNEL SVC BLOCK
            *-----
004E 04BB'         DATA LGCOP,ERRCOP      LOG/ERROR MSG POINTER
0052 0000   COPEN  DATA 0                 OPCODE, RETURN CODE
0054   00          BYTE >00               SUB-OPCODE
0055   1C          BYTE >1C.              LUNO OF CHANNEL
0056   00          BYTE 0                 SYS FLAGS
0057   10          BYTE >10               USER FLAGS=0XX1 0XXX
            *                             BITS 3-4= SHARED ACCESS
0058 0000   COPTYP DATA 0                 RETURNED CHANNEL TYPE
005A 0000   COPMML DATA 0                 RETURNED MAX MSG LEN
005C 0000          DATA 0                 RESERVED
            *-----
            *      WRITE (W/REPLY) IPC MASTER/SLAVE CHANNEL SVC BLOCK
            *-----
005E 04EA'         DATA LGCWR,ERRCWR      LOG/ERROR MSG POINTER
0062 0000   CWRPLY DATA 0                 OPCODE, RETURN CODE
0064   0B          BYTE >0B               SUB-OPCODE
0065   1C          BYTE >1C               LUNO OF CHANNEL
0066   00          BYTE 0                 SYS FLAGS
0067   40          BYTE >40               USER FLAGS= 01XX XXX0
            *                             BIT 1= REPLY BLOCK PRESENT
0068 00BE'         DATA CWBUF             CHANNEL WRITE BUFFER POINTER
006A 0000          DATA 0                 RESERVED
006C 0000   CWRWCC DATA $-$               WRITE CHARACTER COUNT
006E 0070'         DATA CWRRBA            REPLY BLOCK ADDRESS
            *
0070        CWRRBA EVEN
0070 00E6'         DATA CRBUF             CHANNEL REPLY BUFFER POINTER
0072 0028   CWRRBL DATA CRBUFE-CRBUF      REPLY BUFFER LENGTH
0074 0000   CWRRCC DATA $-$               ACTUAL REPLY CHARACTER COUNT
            *-----
            *      CLOSE IPC MASTER/SLAVE CHANNEL SVC BLOCK
            *-----
0076 0519'         DATA LGCCL,ERRCCL      LOG/ERROR MSG POINTER
007A 0000   CCLOSE DATA 0                 OPCODE, RETURN CODE
007C   01          BYTE >01               SUB-OPCODE
007D   1C          BYTE >1C               LUNO OF CHANNEL
007E   00          BYTE 0                 SYS FLAGS
007F   00          BYTE >00               USER FLAGS= 0XXX XXXX
0080 0000          DATA 0,0,0             RESERVED
```

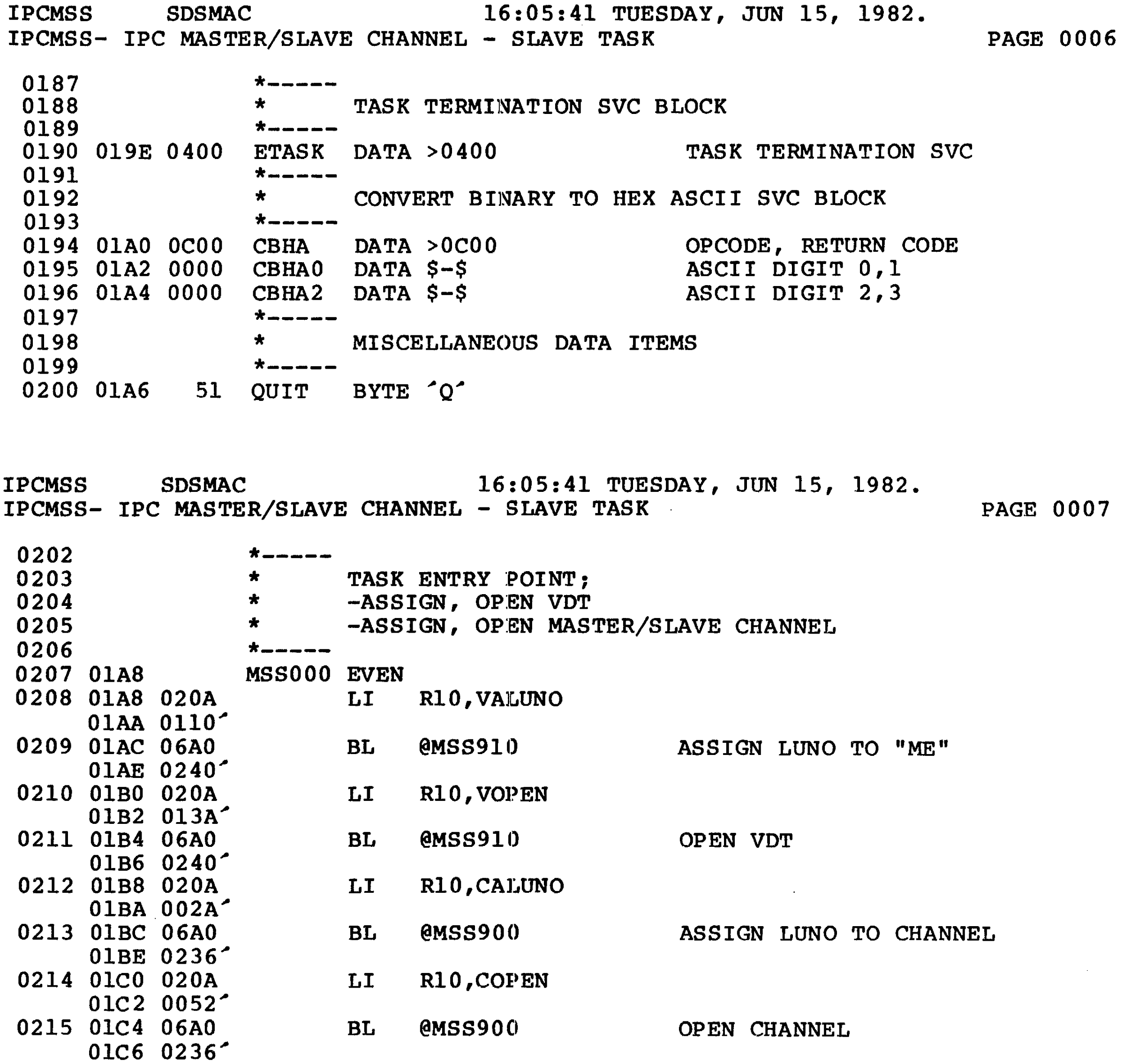
Figure C-2. Slave Task (Sheet 2 of 12)

```
IPCMSS     SDSMAC                  16:05:41 TUESDAY, JUN 15, 1982.
IPCMSS- IPC MASTER/SLAVE CHANNEL - SLAVE TASK                     PAGE 0004

           *-----
           *      RELEASE LUNO FROM IPC MASTER/SLAVE CHANNEL
           *-----
0086 0548'        DATA LGCRL,ERRCRL       LOG/ERROR MSG POINTER
008A 0000  CRLUNO DATA 0                  OPCODE, RETURN CODE
008C   93         BYTE >93                SUB-OPCODE
008D   1C         BYTE >1C                LUNO OF CHANNEL
008E 0000         DATA 0,0,0,0,0,0        RESERVED
009A 0000         DATA >0000              UTILITY FLAGS= XXX0 0XXX
009C 0000         DATA 0,0                RESERVED
00A0 0000         DATA 0                  RESERVED
00A2 0000         DATA 0,0,0,0,0,0        RESERVED
           *-----
           *      IPC MASTER/SLAVE CHANNEL - PATHNAME
           *-----
00AE   0F  CHPTHN BYTE CNEND-$            CHANNEL PATHNAME LENGTH
00AF   56         TEXT 'VOL1.IPC.MSCHAN'    CHANNEL PATHNAME
     00BD' CNEND  EQU  $-1                CHANNEL PATHNAME END
           *-----
           *      IPC MASTER/SLAVE CHANNEL - WRITE BUFFER
           *-----
00BE       CWBUF  EVEN
00BE              BSS  40
     00E6' CWBUFE EQU  $
           *-----
           *      IPC MASTER/SLAVE CHANNEL - REPLY BUFFER
           *-----
00E6       CRBUF  EVEN
00E6              BSS  40
     010E' CRBUFE EQU  $
```

Figure C-2. Slave Task (Sheet 3 of 12)

```
IPCMSS     SDSMAC                 16:05:41 TUESDAY, JUN 15, 1982.
IPCMSS- IPC MASTER/SLAVE CHANNEL - SLAVE TASK                          PAGE 0005

 0130                *-----
 0131                *      ASSIGN LUNO TO VDT SVC BLOCK
 0132                *-----
 0133 010E 0378'            DATA ERRVAL               ERROR MSG POINTER
 0134 0110 0000      VALUNO DATA 0,>911D              ASSIGN LUNO >1D TO VDT
 0135 0114 0000             DATA 0,0,0,0,0,0          RESERVED
 0136 0120 0000             DATA 0,0,0                RESERVED
 0137 0126 0134'            DATA VPATH                VDT PATHNAME POINTER
 0138 0128 0000             DATA 0,0,0,0,0,0          RESERVED
 0139 0134   02      VPATH  BYTE VPATHE-$             PATHNAME LENGTH
 0140 0135   4D             TEXT 'ME'
 0141      0136' VPATHE EQU  $-1                 PATHNAME END POINTER
 0142                *-----
 0143                *      OPEN, REWIND VDT SVC BLOCK
 0144                *-----
 0145 0138 03A6'            DATA ERRVOP               ERROR MSG POINTER
 0146 013A 0000      VOPEN  DATA 0,>031D              OPEN, REWIND LUNO >1D
 0147 013E 0000             DATA 0,0,0,0              RESERVED
 0148                *-----
 0149                *      WRITE TO VDT SVC BLOCK
 0150                *-----
 0151 0146 03D4'            DATA ERRVW                ERROR MSG POINTER
 0152 0148 0000      VWRITE DATA 0,>0B1D              WRITE ASCII TO LUNO >1D
 0153 014C 0000             DATA 0
 0154 014E 0000      VWBUF  DATA $-$                  WRITE BUFFER POINTER
 0155 0150 0000             DATA 0                    RESERVED
 0156 0152 0000      VWLEN  DATA $-$                  WRITE BUFFER LENGTH
 0157                *-----
 0158                *      WRITE (WITH REPLY) VDT SVC BLOCK
 0159                *-----
 0160 0154 0402'            DATA ERRVWR               ERROR MSG POINTER
 0161 0156 0000      VWRPLY DATA 0,>0B1D              WRITE ASCII TO LUNO >1D
 0162 015A 0040             DATA >0040                  WITH REPLY
 0163 015C 059E'            DATA LGWRBF               WRITE BUFFER POINTER
 0164 015E 0000             DATA 0                    RESERVED
 0165 0160 001F             DATA LGWRBE-LGWRBF        WRITE BUFFER LENGTH
 0166 0162 0164'            DATA VWRRBA               REPLY BLOCK ADDRESS
 0167                *
 0168 0164           VWRRBA EVEN
 0169 0164 0000      VWRRBF DATA $-$                  REPLY BUFFER POINTER
 0170 0166 0000      VWRRLN DATA $-$                  REPLY BUFFER LENGTH
 0171 0168 0000      VWRRCC DATA $-$                  REPLY ACTUAL COUNT
 0172                *-----
 0173                *      CLOSE VDT SVC BLOCK
 0174                *-----
 0175 016A 0430'            DATA ERRVCL               ERROR MSG POINTER
 0176 016C 0000      VCLOSE DATA 0,>011D              CLOSE LUNO >1D
 0177 0170 0000             DATA 0,0,0,0              RESERVED
 0178                *-----
 0179                *      RELEASE VDT LUNO SVC BLOCK
 0180                *-----
 0181 0178 045E'            DATA ERRVRL               ERROR MSG POINTER
 0182 017A 0000      VRLUNO DATA 0,>931D              RELEASE LUNO >1D
 0183 017E 0000             DATA 0,0,0,0,0,0          RESERVED
 0184 018A 0000             DATA 0,0,0,0              RESERVED
 0185 0192 0000             DATA 0,0,0,0,0,0          RESERVED
```

Figure C-2. Slave Task (Sheet 4 of 12)



```
IPCMSS     SDSMAC                 16:05:41 TUESDAY, JUN 15, 1982.
IPCMSS- IPC MASTER/SLAVE CHANNEL - SLAVE TASK                      PAGE 0006

 0187                *-----
 0188                *      TASK TERMINATION SVC BLOCK
 0189                *-----
 0190 019E 0400  ETASK  DATA >0400            TASK TERMINATION SVC
 0191                *-----
 0192                *      CONVERT BINARY TO HEX ASCII SVC BLOCK
 0193                *-----
 0194 01A0 0C00  CBHA   DATA >0C00            OPCODE, RETURN CODE
 0195 01A2 0000  CBHA0  DATA $-$              ASCII DIGIT 0,1
 0196 01A4 0000  CBHA2  DATA $-$              ASCII DIGIT 2,3
 0197                *-----
 0198                *      MISCELLANEOUS DATA ITEMS
 0199                *-----
 0200 01A6   51  QUIT   BYTE 'Q'
```

```
IPCMSS     SDSMAC                 16:05:41 TUESDAY, JUN 15, 1982.
IPCMSS- IPC MASTER/SLAVE CHANNEL - SLAVE TASK                      PAGE 0007

 0202                *-----
 0203                *      TASK ENTRY POINT;
 0204                *      -ASSIGN, OPEN VDT
 0205                *      -ASSIGN, OPEN MASTER/SLAVE CHANNEL
 0206                *-----
 0207 01A8           MSS000 EVEN
 0208 01A8 020A             LI   R10,VALUNO
      01AA 0110'
 0209 01AC 06A0             BL   @MSS910         ASSIGN LUNO TO "ME"
      01AE 0240'
 0210 01B0 020A             LI   R10,VOPEN
      01B2 013A'
 0211 01B4 06A0             BL   @MSS910         OPEN VDT
      01B6 0240'
 0212 01B8 020A             LI   R10,CALUNO
      01BA 002A'
 0213 01BC 06A0             BL   @MSS900         ASSIGN LUNO TO CHANNEL
      01BE 0236'
 0214 01C0 020A             LI   R10,COPEN
      01C2 0052'
 0215 01C4 06A0             BL   @MSS900         OPEN CHANNEL
      01C6 0236'
```

Figure C-2. Slave Task (Sheet 5 of 12)

```
IPCMSS     SDSMAC                  16:05:41 TUESDAY, JUN 15, 1982.
IPCMSS- IPC MASTER/SLAVE CHANNEL - SLAVE TASK                        PAGE 0008

 0217                *-----
 0218                *     ISSUE VDT WRITE W/REPLY; ACCEPT MSG TO
 0219                *     TO BE WRITTEN TO CHANNEL
 0220                *----
 0221 01C8 0200  MSS100 LI    R0,CWBUF          SET REPLY BUFFER
      01CA 00BE'
 0222 01CC C800         MOV   R0,@VWRRBF          POINTER IN VDT SVC BLOCK
      01CE 0164'
 0223 01D0 0200         LI    R0,CWBUFE-CWBUF   SET REPLY LENGTH
      01D2 0028
 0224 01D4 C800         MOV   R0,@VWRRLN          IN VDT SVC BLOCK
      01D6 0166'
 0225 01D8 020A         LI    R10,@VWRPLY
      01DA 0156'
 0226 01DC 06A0         BL    @MSS910           WRITE VDT WITH REPLY
      01DE 0240'
 0227 01E0 C820         MOV   @VWRRCC,@CWRWCC   SET CHANNEL WRITE CHAR COUNT
      01E2 0168'
      01E4 006C'
 0228 01E6 020A         LI    R10,CWRPLY
      01E8 0062'
 0229 01EA 06A0         BL    @MSS900           WRITE TO CHANNEL WITH REPLY
      01EC 0236'
 0230 01EE 0209         LI    R9,LGWD           POINT TO LOG MSG
      01F0 0577'
 0231 01F2 06A0         BL    @MSS920           LOG "REPLY" MSG
      01F4 024E'
 0232 01F6 0200         LI    R0,CRBUF
      01F8 00E6'
 0233 01FA C800         MOV   R0,@VWBUF         SET WRITE BUFFER POINTER
      01FC 014E'
 0234 01FE C820         MOV   @CWRRCC,@VWLEN    SET WRITE LENGTH
      0200 0074'
      0202 0152'
 0235 0204 2FE0         SVC   @VWRITE           LOG SLAVE WRITE BUFFER
      0206 0148'
 0236 0208 9820         CB    @CRBUF,@QUIT      DID MASTER REPLY "QUIT"?
      020A 00E6'
      020C 01A6'
 0237 020E 1301         JEQ   MSS700            YES
 0238 0210 10DB         JMP   MSS100            NO, GO GET NEXT MSG FOR MASTER
```

Figure C-2. Slave Task (Sheet 6 of 12)

```
IPCMSS     SDSMAC                   16:05:41 TUESDAY, JUN 15, 1982.
IPCMSS- IPC MASTER/SLAVE CHANNEL - SLAVE TASK                    PAGE 0009

 0240              *----
 0241              *      END SLAVE TASK
 0242              *-----
 0243 0212 020A  MSS700 LI   R10,CCLOSE
      0214 007A'
 0244 0216 06A0         BL   @MSS900            CLOSE MASTER/SLAVE CHANNEL
      0218 0236'
 0245 021A 020A         LI   R10,CRLUNO
      021C 008A'
 0246 021E 06A0         BL   @MSS900            RELEASE LUNO TO CHANNEL
      0220 0236'
 0247 0222 020A         LI   R10,VCLOSE
      0224 016C'
 0248 0226 06A0         BL   @MSS910            CLOSE VDT
      0228 0240'
 0249 022A 020A         LI   R10,VRLUNO
      022C 017A'
 0250 022E 06A0         BL   @MSS910            RELEASE VDT LUNO
      0230 0240'
 0251 0232 2FE0         SVC  @ETASK             TERMINATE TASK
      0234 019E'
```

Figure C-2. Slave Task (Sheet 7 of 12)

```
IPCMSS     SDSMAC                   16:05:41 TUESDAY, JUN 15, 1982.
IPCMSS- IPC MASTER/SLAVE CHANNEL - SLAVE TASK                        PAGE 0010

          *-----
          *      ISSUE SVC AND CHECK FOR ERROR UPON COMPLETION
          *      ENTRY POINTS: MSS900 DO I/O AND LOG TO VDT
          *                    MSS910 DO I/O ONLY (NO LOGGING)
          *                    MSS920 LOG ONLY (NO I/O)
          *-----
0236 C24A MSS900 MOV  R10,R9              POINT TO
0238 0229        AI   R9,-4                 MESSAGE TO BE LOGGED
023A FFFC
023C C259        MOV  *R9,R9                  UPON SUCCESSFUL COMPLETION
023E 1001        JMP  MSS915
0240 04C9 MSS910 CLR  R9                  SET DUMMY LOG POINTER
0242 2FDA MSS915 SVC  *R10                ISSUE SVC
0244 D02A        MOVB @1(R10),R0          GET RETURN CODE; ANY ERROR?
0246 0001
0248 160B        JNE  MSS960              YES, GO SEE IF ERR MSG NEEDED
024A C249        MOV  R9,R9               MSG TO BE LOGGED?
024C 1308        JEQ  MSS950              NO
          *-----
          *      LOG THE I/O (JUST COMPLETED) ON THE VDT
          *-----
024E D039 MSS920 MOVB *R9+,R0             GET MSG LENGTH
0250 0980        SRL  R0,8                SHIFT TO LSB
0252 C800        MOV  R0,@VWLEN           SET WRITE LENGTH
0254 0152'
0256 C809        MOV  R9,@VWBUF           SET BUFFER POINTER
0258 014E'
025A 2FE0        SVC  @VWRITE             LOG MSG TO VDT
025C 0148'
025E 045B MSS950 RT                       NO, EXIT
0260 C06A MSS960 MOV  @-2(R10),R1         IS ERROR MSG NEEDED?
0262 FFFE
0264 13FC        JEQ  MSS950              NO, EXIT
0266 0980        SRL  R0,8                SHIFT ERROR CODE TO LSB
0268 2FE0        SVC  @CBHA               CONVERT TO HEX ASCII
026A 01A0'
026C D020        MOVB @CBHA+1,R0          CONVERT OK?
026E 01A1'
0270 1606        JNE  MSS990              NO, DON'T PLUG INTO MSG
0272 C860        MOV  @CBHA0,@10(R1)      MOVE HEX ERR
0274 01A2'
0276 000A
0278 C860        MOV  @CBHA2,@12(R1)        TO ERROR MSG
027A 01A4'
027C 000C
027E C801 MSS990 MOV  R1,@VWBUF           SET BUFFER POINTER
0280 014E'
0282 0201        LI   R1,ERRLEN           SET ERROR
0284 002D
0286 C801        MOV  R1,@VWLEN             MESSAGE LENGTH
0288 0152'
028A 2FE0        SVC  @VWRITE             LOG ERROR MSG ON VDT
028C 0148'
028E 2FE0        SVC  @ETASK              TERMINATE TASK
0290 019E'
```

Figure C-2. Slave Task (Sheet 8 of 12)

```
 IPCMSS     SDSMAC                  16:05:41 TUESDAY, JUN 15, 1982.
 IPCMSS- IPC MASTER/SLAVE CHANNEL - SLAVE TASK                         PAGE 0011

  0292                *-----
  0293                *      I/O ERROR MESSAGES
  0294                *-----
  0295 0292 0D0A   ERRCAL DATA >0D0A
  0296 0294   49          TEXT 'IPCMSS- XXXX - ERROR ON ASSIGN CHANNEL      '
  0297        002D   ERRLEN EQU  $-ERRCAL-1
  0298 02C0 0D0A   ERRCOP DATA >0D0A
  0299 02C2   49          TEXT 'IPCMSS- XXXX - ERROR ON OPEN CHANNEL        '
  0300 02EE 0D0A   ERRCWR DATA >0D0A
  0301 02F0   49          TEXT 'IPCMSS- XXXX - ERROR ON WRITE CHANNEL W/RPLY'
  0302 031C 0D0A   ERRCCL DATA >0D0A
  0303 031E   49          TEXT 'IPCMSS- XXXX - ERROR ON CLOSE CHANNEL       '
  0304 034A 0D0A   ERRCRL DATA >0D0A
  0305 034C   49          TEXT 'IPCMSS- XXXX - ERROR ON RELEASE CHANNEL LUNO'
  0306                *
  0307 0378 0D0A   ERRVAL DATA >0D0A
  0308 037A   49          TEXT 'IPCMSS- XXXX - ERROR ON ASSIGN VDT          '
  0309 03A6 0D0A   ERRVOP DATA >0D0A
  0310 03A8   49          TEXT 'IPCMSS- XXXX - ERROR ON OPEN VDT            '
  0311 03D4 0D0A   ERRVW  DATA >0D0A
  0312 03D6   49          TEXT 'IPCMSS- XXXX - ERROR ON WRITE VDT           '
  0313 0402 0D0A   ERRVWR DATA >0D0A
  0314 0404   49          TEXT 'IPCMSS- XXXX - ERROR ON WRITE VDT W/REPLY   '
  0315 0430 0D0A   ERRVCL DATA >0D0A
  0316 0432   49          TEXT 'IPCMSS- XXXX - ERROR ON CLOSE VDT           '
  0317 045E 0D0A   ERRVRL DATA >0D0A
  0318 0460   49          TEXT 'IPCMSS- XXXX - ERROR ON RELEASE VDT LUNO    '
```

Figure C-2. Slave Task (Sheet 9 of 12)

```
IPCMSS     SDSMAC               16:05:41 TUESDAY, JUN 15, 1982.
IPCMSS- IPC MASTER/SLAVE CHANNEL - SLAVE TASK                    PAGE 0012
             *-----
             *      CHANNEL I/O ACTIVITY MESSAGES LOGGED TO VDT
             *-----
048C    2E   LGCAL  BYTE LGCALE-$
048D    0D          BYTE >0D,>0A
048F    49          TEXT 'IPCMSS-001 IPC MASTER/SLAVE CHANNEL ASSIGNED'
       04BA' LGCALE EQU  $-1
             *
04BB    2E   LGCOP  BYTE LGCOPE-$
04BC    0D          BYTE >0D,>0A
04BE    49          TEXT 'IPCMSS-002 IPC MASTER/SLAVE CHANNEL OPENED   '
       04E9' LGCOPE EQU  $-1
             *
04EA    2E   LGCWR  BYTE LGCWRE-$
04EB    0D          BYTE >0D,>0A
04ED    49          TEXT 'IPCMSS-003 CHANNEL WRITE W/REPLY ISSUED      '
       0518' LGCWRE EQU  $-1
             *
0519    2E   LGCCL  BYTE LGCCLE-$
051A    0D          BYTE >0D,>0A
051C    49          TEXT 'IPCMSS-004 IPC MASTER/SLAVE CHANNEL CLOSED   '
       0547' LGCCLE EQU  $-1
             *
0548    2E   LGCRL  BYTE LGCRLE-$
0549    0D          BYTE >0D,>0A
054B    49          TEXT 'IPCMSS-005 IPC MASTER/SLAVE CHANNEL RELEASED'
       0576' LGCRLE EQU  $-1
             *
0577    26   LGWD   BYTE LGWDE-$
0578    0D          BYTE >0D,>0A
057A    49          TEXT 'IPCMSS-006 DATA REPLY FROM MASTER:  '
       059D' LGWDE  EQU  $-1
             *
059E  0A0D   LGWRBF DATA >0A0D
05A0    49          TEXT 'IPCMSS-007 ENTER WRITE DATA:  '
       05BD' LGWRBE EQU  $-1
                    END
NO ERRORS,     NO WARNINGS
```

Figure C-2. Slave Task (Sheet 10 of 12)

```
IPCMSS    SDSMAC                 16:05:41 TUESDAY, JUN 15, 1982.
LABEL     VALUE   DEFN  REFERENCES                                            PAGE 0013

$         05BE'         0081  0081  0087  0087  0114  0116  0122  0128  0139
                        0141  0154  0154  0156  0156  0169  0169  0170  0170
                        0171  0171  0195  0195  0196  0196  0297  0323  0326
                        0328  0331  0333  0336  0338  0341  0343  0346  0348
                        0351  0355
CALUNO    002A'   0047  0212
CBHA      01A0'   0194  0281  0282
CBHA0     01A2'   0195  0284
CBHA2     01A4'   0196  0285
CCLOSE    007A'   0092  0243
CHPTHN    00AE'   0114  0054
CNEND     00BD'   0116  0114
COPEN     0052'   0060  0214
COPMML    005A'   0067
COPTYP    0058'   0066
CRBUF     00E6'   0126  0085  0086  0232  0236
CRBUFE    010E'   0128  0086
CRLUNO    008A'   0103  0245
CWBUF     00BE'   0120  0079  0221  0223
CWBUFE    00E6'   0122  0223
CWRPLY    0062'   0073  0228
CWRRBA    0070'   0084  0082
CWRRBL    0072'   0086
CWRRCC    0074'   0087  0234
CWRWCC    006C'   0081  0227
ERRCAL    0292'   0295  0046  0297
ERRCCL    031C'   0302  0091
ERRCOP    02C0'   0298  0059
ERRCRL    034A'   0304  0102
ERRCWR    02EE'   0300  0072
ERRLEN    002D    0297  0287
ERRVAL    0378'   0307  0133
ERRVCL    0430'   0315  0175
ERRVOP    03A6'   0309  0145
ERRVRL    045E'   0317  0181
ERRVW     03D4'   0311  0151
ERRVWR    0402'   0313  0160
ETASK     019E'   0190  0251  0290
LGCAL     048C'   0323  0046
LGCALE    04BA'   0326  0323
LGCCL     0519'   0338  0091
LGCCLE    0547'   0341  0338
LGCOP     04BB'   0328  0059
LGCOPE    04E9'   0331  0328
LGCRL     0548'   0343  0102
LGCRLE    0576'   0346  0343
LGCWR     04EA'   0333  0072
LGCWRE    0518'   0336  0333
LGWD      0577'   0348  0230
LGWDE     059D'   0351  0348
LGWRBE    05BD'   0355  0165
LGWRBF    059E'   0353  0163  0165
MSS000    01A8'   0207  0038
MSS100    01C8'   0221  0238
MSS700    0212'   0243  0237
MSS900    0236'   0259  0213  0215  0229  0244  0246
MSS910    0240'   0263  0209  0211  0226  0248  0250
MSS915    0242'   0264  0262
MSS920    024E'   0272  0231
MSS950    025E'   0277  0268  0279
```

Figure C-2. Slave Task (Sheet 11 of 12)

```
IPCMSS     SDSMAC                  16:05:41 TUESDAY, JUN 15, 1982.
LABEL    VALUE   DEFN  REFERENCES                                        PAGE 0014

MSS960    0260'   0278  0266
MSS990    027E'   0286  0283
MSSWP     0006'   0039  0038
QUIT      01A6'   0200  0236
R0        0000          0221  0222  0223  0224  0232  0233  0265  0272  0273
                        0274  0280  0282
R1        0001          0278  0284  0285  0286  0287  0288
R10       000A          0208  0210  0212  0214  0225  0228  0243  0245  0247
                        0249  0259  0264  0265  0278
R9        0009          0230  0259  0260  0261  0261  0263  0267  0267  0272
                        0275
SVC               0041
VALUNO    0110'   0134  0208
VCLOSE    016C'   0176  0247
VOPEN     013A'   0146  0210
VPATH     0134'   0139  0137
VPATHE    0136'   0141  0139
VRLUNO    017A'   0182  0249
VWBUF     014E'   0154  0233  0275  0286
VWLEN     0152'   0156  0234  0274  0288
VWRITE    0148'   0152  0235  0276  0289
VWRPLY    0156'   0161  0225
VWRRBA    0164'   0168  0166
VWRRBF    0164'   0169  0222
VWRRCC    0168'   0171  0227
VWRRLN    0166'   0170  0224
```

Figure C-2. Slave Task (Sheet 12 of 12)

Alphabetical Index

Introduction

HOW TO USE INDEX

The index, table of contents, list of illustrations, and list of tables are used in conjunction to obtain the location of the desired subject. Once the subject or topic has been located in the index, use the appropriate paragraph number, figure number, or table number to obtain the corresponding page number from the table of contents, list of illustrations, or list of tables.

INDEX ENTRIES

The following index lists key words and concepts from the subject material of the manual together with the area(s) in the manual that supply major coverage of the listed concept. The numbers along the right side of the listing reference the following manual areas:

- Sections — Reference to Sections of the manual appear as "Sections x" with the symbol x representing any numeric quantity.

- Appendixes — Reference to Appendixes of the manual appear as "Appendix y" with the symbol y representing any capital letter.

- Paragraphs — Reference to paragraphs of the manual appear as a series of alphanumeric or numeric characters punctuated with decimal points. Only the first character of the string may be a letter; all subsequent characters are numbers. The first character refers to the section or appendix of the manual in which the paragraph may be found.

- Tables — References to tables in the manual are represented by the capital letter T followed immediately by another alphanumeric character (representing the section or appendix of the manual containing the table). The second character is followed by a dash (-) and a number.

  Tx-yy

- Figures — References to figures in the manual are represented by the capital letter F followed immediately by another alphanumeric character (representing the section or appendix of the manual containing the figure). The second character is followed by a dash (-) and a number.

  Fx-yy

- Other entries in the Index — References to other entries in the index preceded by the word "See" followed by the referenced entry.

USER'S RESPONSE SHEET

Manual Title: DNOS Supervisor Call (SVC) Reference Manual (2270507-9701)

Manual Date: March 1985 Date of This Letter: ________

User's Name: ________ Telephone: ________

Company: ________ Office/Department: ________

Street Address: ________

City/State/Zip Code: ________

Please list any discrepancy found in this manual by page, paragraph, figure, or table number in the following space. If there are any other suggestions that you wish to make, feel free to include them. Thank you.

Location in Manual **Comment/Suggestion**

NO POSTAGE NECESSARY IF MAILED IN U.S.A.
FOLD ON TWO LINES (LOCATED ON REVERSE SIDE), TAPE AND MAIL

CUT ALONG LINE

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 7284 DALLAS, TX

POSTAGE WILL BE PAID BY ADDRESSEE

TEXAS INSTRUMENTS INCORPORATED
DATA SYSTEMS GROUP

ATTN: TECHNICAL PUBLICATIONS
P.O. Box 2909 M/S 2146
Austin, Texas 78769

FOLD

Appendix A

SVC Index

This appendix contains an index of supervisor calls (SVCs). Several SVCs consist of operations designated by sub-opcodes. Both SVCs and operations appear in the index. The I/O resources are also listed in the index to direct you to the descriptions of the operations for each resource.

SVC **Paragraph**