

DNOS



Systems Programmer's Guide

Part No. 2270510-9701 *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INSERT LATEST CHANGED PAGES AND DISCARD SUPERSEDED PAGES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 Systems Programmer's Guide (2270510-9701 *C)

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

Total number of pages in this publication is 374 consisting of the following:

PAGE NO.	CHANGE NO.	PAGE NO.	CHANGE NO.	PAGE NO.	CHANGE NO.
Cover	1	2-9 - 2-10	1	3-91	1
Effective Pages	1	2-11 - 2-56	0	3-92	0
Eff. Pages (Cont.)	1	3-1 - 3-11	0	4-1 - 4-6	0
iii - iv	1	3-12	1	5-1 - 5-15	0
v - vii	0	3-13 - 3-29	0	5-16	1
viii	1	3-30	1	5-17 - 5-68	0
ix - xviii	0	3-31 - 3-32	0	6-1	1
1-1 - 1-5	0	3-33	1	6-2 - 6-8	0
1-6	1	3-34 - 3-59	0	7-1 - 7-10	0
1-7 - 1-12	0	3-60	1	8-1 - 8-12	0
1-13 - 1-14	1	3-61 - 3-86	0	9-1 - 9-8	0
1-15 - 1-28	0	3-87	1	9-9	1
2-1 - 2-8	0	3-88 - 3-90	0	9-10 - 9-12	0

The computers, 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.

© 1981, 1982, 1983, 1985, Texas Instruments Incorporated. All Rights Reserved.

Printed in U.S.A.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of Texas Instruments Incorporated.



Manual Update

MANUAL: DNOS Systems Programmer's Guide (2270510-9701 *C)
MCR/CHANGE NO.: MCR 004724/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**

Cover/Manual Revision History
iii - iv
vii - viii
1-5 - 1-6
1-13 - 1-14
2-9 - 2-10
3-11 - 3-12
3-29 - 3-30
3-33 - 3-34
3-59 - 3-60
3-87 - 3-88
3-91 - 3-92
5-15 - 5-16
6-1 - 6-2
9-9 - 9-10
User's Resp./Bus. Reply
Inside Cover/Cover

**Insert
Change 1 Pages**

Cover/Effective Pages
iii - iv
vii - viii
1-5 - 1-6
1-13 - 1-14
2-9 - 2-10
3-11 - 3-12
3-29 - 3-30
3-33 - 3-34
3-59 - 3-60
3-87 - 3-88
3-91 - 3-92
5-15 - 5-16
6-1 - 6-2
9-9 - 9-10
User's Resp./Bus. Reply
Inside Cover/Cover



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 Systems Programmer's Guide (2270510-9701 *C)

Continued:

PAGE NO.	CHANGE NO.	PAGE NO.	CHANGE NO.	PAGE NO.	CHANGE NO.
10-1 - 10-80				
11-1 - 11-120				
12-1 - 12-140				
A-1 - A-140				
Index-1 - Index-100				
User's Response1				
Business Reply1				
Inside Cover1				
Cover1				



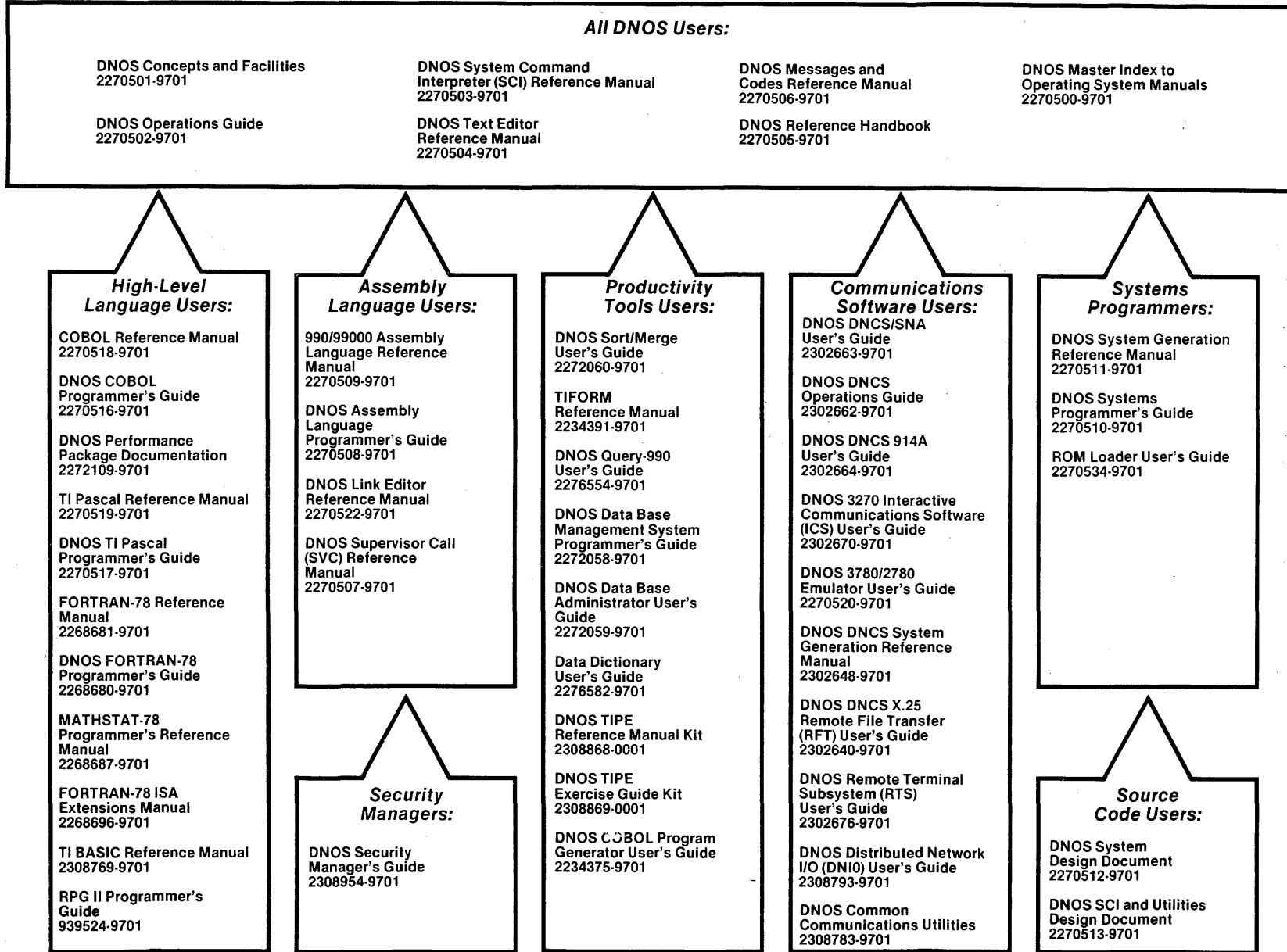
DNOS Software Manuals

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.

2270510-9701

Change 1

iii



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 discusses the functional characteristics of the Texas Instruments Distributed Network Operating System (DNOS) and is intended for the systems programmer. It provides information required for modifying the DNOS system to meet the specific needs of an applications environment. Directed to object-level implementation, this manual is not a detailed guide for the user who has access to source code.

This manual contains the following sections and appendixes:

Section

- 1 **DNOS System Overview** — Discusses the structure, flow, and loading of the system and the distinctive DNOS subsystems. The subsystems described are job management, segment management, name management, and interprocess communication (IPC).
- 2 **Disk and File Organization** — Describes the files that DNOS supports and the disk structures that support the files.
- 3 **Extending SCI** — Describes the System Command Interpreter (SCI), the use of SCI primitives, and the procedure for adding commands to support a specific applications environment.
- 4 **Writing an SVC Processor** — Explains the procedure for writing supervisor call (SVC) processors for applications-oriented SVCs and for including these processors in a custom-generated system.
- 5 **Writing a DSR** — Describes the input/output (I/O) system and how to write a device service routine (DSR). This section also describes the interrupt types and the DSR support routines.
- 6 **DNOS Accounting System** — Discusses the capabilities and use of the accounting system.
- 7 **File Security** — Explains what a programmer needs to know about a system that supports file security.
- 8 **Analyzing System Problems** — Describes system crashes that occur during system loading and after the system begins executing. This section discusses both the system crash dump and utility XANAL, which lists the data in the dump. It also explains how to force a system crash.

Section

- 9 Adding Error Messages — Discusses error processing and explains how to add error messages required for user-supplied system programs.
- 10 International Considerations — Explains the method and results of changing the country code of the system and how to customize DNOS for a particular country.
- 11 Differences Between DX10 and DNOS — Describes the differences between DX10 and DNOS as they relate to migration of DX10 to DNOS.
- 12 Special Features of DNOS — Describes some special features of DNOS and how they are used.

The DNOS software manuals shown on the support manual diagram (frontispiece) contain related information. The *ROM Loader User's Guide* (part number 2270534-9701) contains information about how to load DNOS from devices accessible on the TILINE* peripheral bus and on the communications register unit (CRU).

* TILINE is a registered trademark of Texas Instruments Incorporated.

Contents

Paragraph	Title	Page
1 — DNOS System Overview		
1.1	DNOS System Structure	1-1
1.1.1	System Memory Mapping	1-3
1.1.2	Fundamental Structure of DNOS	1-5
1.1.2.1	Job Structure	1-5
1.1.2.2	Tasks	1-5
1.1.2.3	Queue Servers	1-6
1.1.2.4	System Files	1-6
1.2	Channels for DNOS Functions	1-9
1.3	System Flow	1-10
1.3.1	Initial Program Load (IPL)	1-11
1.3.2	System Idle State	1-11
1.3.3	Task Activation	1-11
1.3.4	Task Scheduling	1-11
1.3.5	Time Slicing	1-11
1.3.6	XOP Processing	1-11
1.3.7	Execution Priorities	1-12
1.3.8	Dynamic Modification of Run-Time Parameters	1-13
1.3.9	Task Termination	1-14
1.3.10	Clock Interrupt Processor	1-14
1.3.11	Internal Interrupt Processor	1-14
1.3.12	System Crash Routine	1-15
1.4	IPL and System Loaders	1-15
1.4.1	ROM Loader	1-16
1.4.2	Program Image Loader	1-16
1.4.3	System Loader	1-17
1.5	DNOS Subsystems	1-18
1.5.1	Job Management	1-18
1.5.1.1	Supervisor Call (SVC)	1-18
1.5.1.2	SCI Commands	1-19
1.5.2	Segment Management	1-19
1.5.3	Name Management	1-21
1.5.3.1	Logical Names	1-21
1.5.3.2	Job Temporary Files	1-21
1.5.3.3	File Concatenation	1-21
1.5.3.4	Multivolume File Capability	1-23
1.5.3.5	Stages of Name Definition	1-23

Paragraph	Title	Page
1.5.4	Interprocess Communication (IPC)	1-24
1.5.4.1	Channel Definition	1-24
1.5.4.2	Channel Creation	1-25
1.5.4.3	Channel Characteristics	1-25
1.5.4.4	IPC Supervisor Calls (SVCs)	1-27
1.5.4.5	IPC SCI Commands	1-27

2 — Disk and File Organization

2.1	File Organization	2-1
2.1.1	File Types	2-1
2.1.1.1	Sequential Files	2-1
2.1.1.2	Relative Record Files	2-2
2.1.1.3	Key Indexed Files (KIFs)	2-2
2.1.1.4	Special Usage Files	2-3
2.1.2	File Protection and Sharing	2-3
2.1.2.1	Delete and Write Protection	2-3
2.1.2.2	Record Locking	2-4
2.1.2.3	Access Privileges	2-4
2.1.2.4	Special Usage File Protection	2-4
2.1.3	File Characteristics	2-5
2.1.3.1	Record Blocking	2-5
2.1.3.2	Saving Disk Space	2-6
2.1.3.3	Immediate Write	2-6
2.1.3.4	Temporary Attribute	2-7
2.1.3.5	Expandability	2-7
2.1.3.6	End-of-File (EOF)	2-7
2.2	Disk Organization	2-8
2.2.1	Disk Characteristics	2-8
2.2.2	Disk Space Allocation to Files	2-9
2.2.3	Physical Organization of a DNOS Disk	2-10
2.2.3.1	Volume Information	2-10
2.2.3.2	Bit Map	2-15
2.2.4	Displaying and Modifying Absolute Disk Addresses	2-15
2.3	Disk File Structures	2-15
2.3.1	Directory File	2-16
2.3.1.1	Directory File Characteristics	2-16
2.3.1.2	File Descriptor Record (FDR)	2-18
2.3.1.3	Alias Descriptor Record (ADR)	2-23
2.3.1.4	Channel Descriptor Record (CDR)	2-25
2.3.1.5	Key Descriptor Record (KDR)	2-30
2.3.1.6	Example of a Dump Directory	2-32
2.3.2	Sequential Files	2-32
2.3.3	Relative Record Files	2-37
2.3.3.1	Unblocked Relative Record Files	2-37
2.3.3.2	Blocked Relative Record Files	2-40
2.3.4	Key Indexed Files (KIFs)	2-40
2.3.4.1	KIF Keys	2-40

Paragraph	Title	Page
2.3.4.2	KIF Records	2-41
2.3.4.3	KIF Key and Record Example	2-41
2.3.4.4	Structure of KIFs	2-42
2.3.4.5	Description of Logical Record	2-51
2.3.4.6	KIF Disk Usage	2-52

3 — Extending SCI

3.1	SCI Overview	3-1
3.2	User-Defined SCI Command Procedures	3-3
3.2.1	SCI Primitive	3-4
3.2.2	Command Procedure	3-4
3.2.3	Command Processor	3-4
3.3	SCI Language Syntax	3-4
3.4	SCI Language Variables	3-6
3.4.1	Synonyms	3-7
3.4.2	Logical Names	3-11
3.4.3	Environment and Scope of Name Definitions	3-11
3.4.4	Field Prompts	3-12
3.4.4.1	ACNM Field Prompt Type	3-13
3.4.4.2	DEFAULT Field Prompt Type	3-14
3.4.4.3	ELEMENT Field Prompt Type	3-14
3.4.4.4	INT Field Prompt Type	3-15
3.4.4.5	NAME Field Prompt Type	3-15
3.4.4.6	RANGE Field Prompt Type	3-15
3.4.4.7	STRING Field Prompt Type	3-15
3.4.4.8	YESNO Field Prompt Type	3-16
3.5	SCI Primitives	3-16
3.5.1	.PROC and .EOP Primitives	3-18
3.5.2	.IF, .ELSE, .ENDIF Primitives	3-20
3.5.3	.PROMPT Primitive	3-22
3.5.4	.SYN Primitive	3-23
3.5.5	.EXIT Primitive	3-24
3.5.6	.EVAL Primitive	3-25
3.5.7	.LOOP, .UNTIL, .WHILE, and .REPEAT Primitives	3-26
3.5.8	.SPLIT Primitive	3-28
3.5.8.1	Using the First .SPLIT Format	3-28
3.5.8.2	Using the Second .SPLIT Format	3-29
3.5.9	.BID Primitive	3-30
3.5.10	.DBID Primitive	3-31
3.5.11	.QBID Primitive	3-32
3.5.12	.RBID Primitive	3-32
3.5.13	.DATA and .EOD Primitives	3-33
3.5.14	.STOP Primitive	3-34
3.5.15	.USE Primitive	3-35
3.5.16	.OPTION Primitive	3-36
3.5.17	.MENU Primitive	3-38
3.5.18	.SHOW Primitive	3-39

Paragraph	Title	Page
3.5.19	.SVC Primitive	3-39
3.6	SCI Primitive Batch Stream Example	3-44
3.7	Error Processing for Primitives	3-46
3.8	Command Procedures and Command Processors	3-46
3.8.1	Command Procedure Design	3-46
3.8.2	Command Processor Design	3-51
3.8.3	Installing Command Procedures and Command Processors	3-53
3.8.4	Using New Commands	3-54
3.8.5	Expert Mode Considerations	3-54
3.8.6	Deleting Commands	3-55
3.9	Command Processor Interface Routines	3-56
3.9.1	Interface Routine References	3-56
3.9.2	Buffers for Interface Routines	3-57
3.10	Interface Routine Descriptions	3-58
3.10.1	SCI Interface Routines	3-58
3.10.1.1	Get TCA (S\$GTCA)	3-58
3.10.1.2	Initialize System Data Base (S\$NEW)	3-58
3.10.1.3	Bid Task Routine (S\$BIDT)	3-59
3.10.1.4	Get Parameter (S\$PARM)	3-62
3.10.1.5	Get Terminal Status (S\$STAT)	3-64
3.10.1.6	Set Synonym Value (S\$SETS)	3-64
3.10.1.7	Get Synonym Value (S\$MAPS)	3-65
3.10.1.8	Search Name Correspondence Table (S\$SNCT)	3-66
3.10.1.9	Split List Into Components (S\$SPLT)	3-67
3.10.1.10	Return Time and Date (S\$TAD)	3-68
3.10.1.11	Put TCA (S\$PTCA)	3-69
3.10.1.12	Release TCA (S\$RTCA)	3-69
3.10.1.13	Create Message (S\$CMMSG)	3-69
3.10.1.14	Terminate and Return to SCI (S\$TERM)	3-72
3.10.1.15	Alternate Termination (S\$STOP)	3-73
3.10.2	Local Display File Routines	3-73
3.10.2.1	Open File (S\$OPEN)	3-74
3.10.2.2	Open File Specifying User ID (S\$OPNS)	3-74
3.10.2.3	Write to File (S\$WRIT)	3-75
3.10.2.4	Write End-of-Line to File (S\$WEOL)	3-75
3.10.2.5	Close File (S\$CLOS)	3-75
3.10.2.6	Local Display File Example	3-76
3.10.3	String Utility Routines	3-77
3.10.3.1	Convert ASCII to Binary Integer (S\$INT)	3-77
3.10.3.2	Convert Binary Integer to ASCII (S\$IASC)	3-78
3.10.3.3	Compare Strings (S\$SCOM)	3-79
3.10.3.4	Copy String (S\$SCPY)	3-80
3.10.4	Arithmetic Utility Routines	3-81
3.10.4.1	Add 32-Bit Integers (S\$IADD)	3-81
3.10.4.2	Subtract 32-Bit Integers (S\$ISUB)	3-82
3.10.4.3	Multiply 32-Bit Integers (S\$IMUL)	3-83
3.10.4.4	Divide 32-Bit Integers (S\$IDIV)	3-84
3.10.5	Spooler Interface Routine (S\$SPLR)	3-85
3.10.6	Operator Interface Routines	3-88

Paragraph	Title	Page
3.10.6.1	Initialize Operator Interface (OI\$BGN)	3-89
3.10.6.2	Create Operator Message (OI\$COM)	3-89
3.10.6.3	Wait for Operator Response (OI\$WAT)	3-90
3.10.6.4	End Operator Interface Subsystem Interface Session (OI\$END)	3-90
3.10.7	Mailbox Subsystem Interface Routines	3-91
3.10.7.1	Initialize Mailbox Interface (MB\$INT)	3-91
3.10.7.2	Send Mail (MB\$SND)	3-91
3.10.7.3	Receive Mail (MB\$RCV)	3-92
3.10.7.4	Release Mailbox (MB\$RLS)	3-92

4 — Writing an SVC Processor

4.1	Need for an SVC Processor	4-1
4.2	How to Write an SVC Processor	4-1
4.2.1	SVC Call Block	4-1
4.2.2	SVC Definition Tables	4-1
4.2.3	SVC Processor Details	4-4
4.2.4	System Generation Requirements	4-6

5 — Writing a DSR

5.1	Introduction	5-1
5.2	Preparation	5-1
5.3	I/O Subsystem	5-2
5.3.1	Data Structures	5-2
5.3.2	Request Flow	5-3
5.3.3	Device Interrupt Decoder	5-4
5.3.3.1	Single-Device Interrupt	5-4
5.3.3.2	Multiple-Device Interrupt	5-5
5.3.3.3	Expansion Chassis Interrupt	5-7
5.3.3.4	Asynchronous Multiplexer Interrupt Decoder	5-10
5.3.3.5	Return Routine	5-13
5.3.3.6	Illegal Interrupt Routine	5-13
5.4	Device Service Routines	5-13
5.4.1	Design Criteria	5-14
5.4.2	DSR Entry Points	5-16
5.4.2.1	Hardware Interrupt Entry Point	5-17
5.4.2.2	Delayed Reentry Point	5-19
5.4.2.3	Power-Up Initialization Entry Point	5-19
5.4.2.4	Abort Entry Point	5-20
5.4.2.5	Time-Out Entry Point	5-20
5.4.2.6	Initial Request Entry Point	5-20
5.4.2.7	Priority Scheduler Entry Point	5-20
5.4.3	Body of the DSR	5-20
5.4.4	Bidding a Task From the DSR	5-21
5.4.5	Multiplexing Hidden Request Queue	5-21
5.5	DSR Support Routines	5-22

Paragraph	Title	Page
5.5.1	Branch Table Processor Routine	5-22
5.5.2	End-of-Record Processor Routine	5-24
5.5.3	Bid Task Routine	5-24
5.5.4	Queue Event Character or Queue Character Routine	5-25
5.5.5	Get Queued Character Routine	5-26
5.5.6	Get Event Character	5-26
5.5.7	Character Check Routines	5-27
5.5.8	Map Out Current Buffer Routine	5-28
5.5.9	Get Buffer	5-28
5.5.10	Map In Buffer	5-29
5.5.11	Get 20-Bit TILINE Address	5-29
5.5.12	Transfer PDT Information to Task Memory	5-30
5.5.13	Report TILINE Error	5-30
5.6	Asynchronous DSR Structure	5-31
5.6.1	Asynchronous DSR Design Overview	5-34
5.6.2	Terminal Service Routine (TSR)	5-36
5.6.3	Interrupt Service Routine (ISR)	5-37
5.6.4	Hardware Controller Service Routine (HSR)	5-38
5.6.4	Asynchronous Data Structure Allocation	5-39
5.7	HSR Common Subroutines	5-41
5.7.1	Power-Up Initialization	5-42
5.7.2	Write Output Signal or Function	5-43
5.7.2.1	HSTBIL Subroutine	5-43
5.7.2.2	HSTCR Subroutine	5-43
5.7.2.3	HSTCTH Subroutine	5-44
5.7.2.4	HSTRS Subroutine	5-44
5.7.2.5	HSTTB Subroutine	5-44
5.7.2.6	HSTUIL Subroutine	5-44
5.7.3	Read Input Signal or Function	5-44
5.7.4	Enable/Disable Status Change Notification	5-45
5.7.5	Output a Character	5-46
5.7.6	Write Operational Parameters	5-46
5.7.6.1	Set Channel Speed (Baud Rate)	5-47
5.7.6.2	Set Data Character Format	5-48
5.7.7	Read Operational Parameters and Information	5-48
5.7.8	Request Time Interval Notification	5-49
5.7.9	Controller Interrupt Decoder	5-50
5.8	DSR Installation	5-51
5.9	Debugging Techniques	5-52
5.10	DSR Example	5-52

6 — DNOS Accounting System

6.1	Introduction	6-1
6.2	Accounting Data	6-1
6.2.1	Description of Accumulated Data	6-2
6.2.2	Data Format	6-2
6.3	Implementation	6-6

Paragraph	Title	Page
6.3.1	Account Numbers	6-7
6.3.2	System Generation Requirements	6-7
6.3.3	Application Program Requirements	6-7

7 — File Security

7.1	Introduction	7-1
7.2	Access Groups	7-1
7.2.1	SYSMGR Access Group Member	7-2
7.2.2	Access Group Leader	7-2
7.2.3	Access Group Member	7-2
7.2.4	Creation Access Group	7-2
7.2.5	Modifications to Access Groups and Access Rights	7-3
7.3	Access Rights	7-3
7.3.1	Control Access	7-4
7.3.2	Read Access	7-4
7.3.3	Write Access	7-4
7.3.4	Execute Access	7-4
7.3.5	Delete Access	7-4
7.4	Example of a Secured System	7-4
7.5	Important Points About Access Rights to Secured Files	7-7
7.6	Programmers	7-8
7.6.1	I/O Utility Operations that Specify a User ID	7-8
7.6.2	Security Bypass	7-9
7.6.3	Special Rename File SVC Option	7-9
7.6.4	Open Routine Specifying User ID (S\$OPNS)	7-10
7.6.5	No-Echo Option for SCI Prompt Response	7-10
7.6.6	Read File Characteristics Option	7-10

8 — Analyzing System Problems

8.1	System Initialization Problems	8-1
8.1.1	ROM Loader Errors	8-1
8.1.2	Program Image Loader Errors	8-2
8.1.3	System Loader Errors	8-2
8.2	System Crash Problems	8-4
8.2.1	Organization of the XANAL Listing	8-6
8.2.1.1	Crash Code	8-6
8.2.1.2	Executing Task	8-6
8.2.1.3	Executing Task JSB	8-6
8.2.1.4	Location of Failure	8-6
8.2.1.5	Status Register	8-6
8.2.1.6	CURMAP Addr	8-6
8.2.1.7	System Patch Area	8-8
8.2.1.8	Executing Workspace at Time of Dump	8-8
8.2.1.9	Hardware Trap and Extended Operation (XOP) Vectors	8-8
8.2.1.10	Special Workspaces	8-8

Paragraph	Title	Page
8.2.2	Task States and JSBs	8-8
8.2.2.1	Task States	8-11
8.2.2.2	JSB List	8-11
8.2.3	Analyzing System Crash Dumps	8-11
8.2.4	Forcing a System Crash	8-12

9 — Adding Error Messages

9.1	DNOS Message and Error Processing	9-1
9.1.1	Internal Error Codes	9-1
9.1.2	Short English Messages	9-2
9.1.3	Expanded Explanations Online	9-3
9.1.4	Show Expanded Message (SEM) Utility	9-3
9.1.5	Displaying Messages	9-3
9.1.6	Message Examples	9-4
9.1.6.1	Assign LUNO Error	9-4
9.1.6.2	COBOL Compiler Termination	9-5
9.2	Message Files	9-5
9.2.1	Format of the Message Text Files	9-6
9.2.2	Format of the Expanded Error Message Text Files	9-8
9.3	Message File Utilities	9-9
9.3.1	Message File Utility	9-10
9.3.2	Expanded Message File Utility	9-11
9.4	Error Message Interface	9-11

10 — International Considerations

10.1	Introduction	10-1
10.2	Country Code	10-2
10.3	Information Interchange Codes	10-3
10.4	KIF Collating Sequences	10-5
10.5	Internationalizing Messages	10-7

11 — Differences Between DX10 and DNOS

11.1	Introduction	11-1
11.2	General Environment	11-1
11.3	Device and File I/O Operations	11-4
11.4	SCI User Interface	11-6
11.5	SCI Primitives and Interface Routines	11-7
11.6	SVC Support	11-8
11.7	User-Written System Software	11-11
11.8	System Console	11-11

Paragraph	Title	Page
12 — Special Features of DNOS		
12.1	Introduction	12-1
12.2	DNOS Job Architecture	12-1
12.3	Logical Names	12-2
12.4	SCI Features	12-3
12.5	Interprocess Communication (IPC)	12-3
12.6	Performance Monitoring	12-4
12.7	Performance Optimization	12-4
12.7.1	Alternate Ways to Structure SCI Sessions	12-5
12.7.2	Keyboard Bidding of Tasks	12-5
12.7.3	Types of LUNOs	12-7
12.7.4	Batch Jobs	12-7
12.7.5	Miscellaneous Items	12-7
12.8	System Configuration Utility	12-8
12.9	S\$SYSTEM Directory	12-8
12.10	System Command Procedures	12-8
12.10.1	Begin Update Documentation (BUD)	12-8
12.10.2	End Update Documentation (EUD)	12-10
12.11	Maintaining User IDs	12-11
12.12	Spooler Subsystem	12-11
12.12.1	Spooler Directory	12-11
12.12.1.1	Spooler Banner Sheet File	12-12
12.12.1.2	Spooler Queue File	12-12
12.12.2	Spooler Device Attributes	12-13
12.12.3	Spooler Clean-up	12-13
12.12.4	Spooler Temporary Files	12-14

Appendixes

Appendix	Title	Page
A	Keycap Cross-Reference	A-1

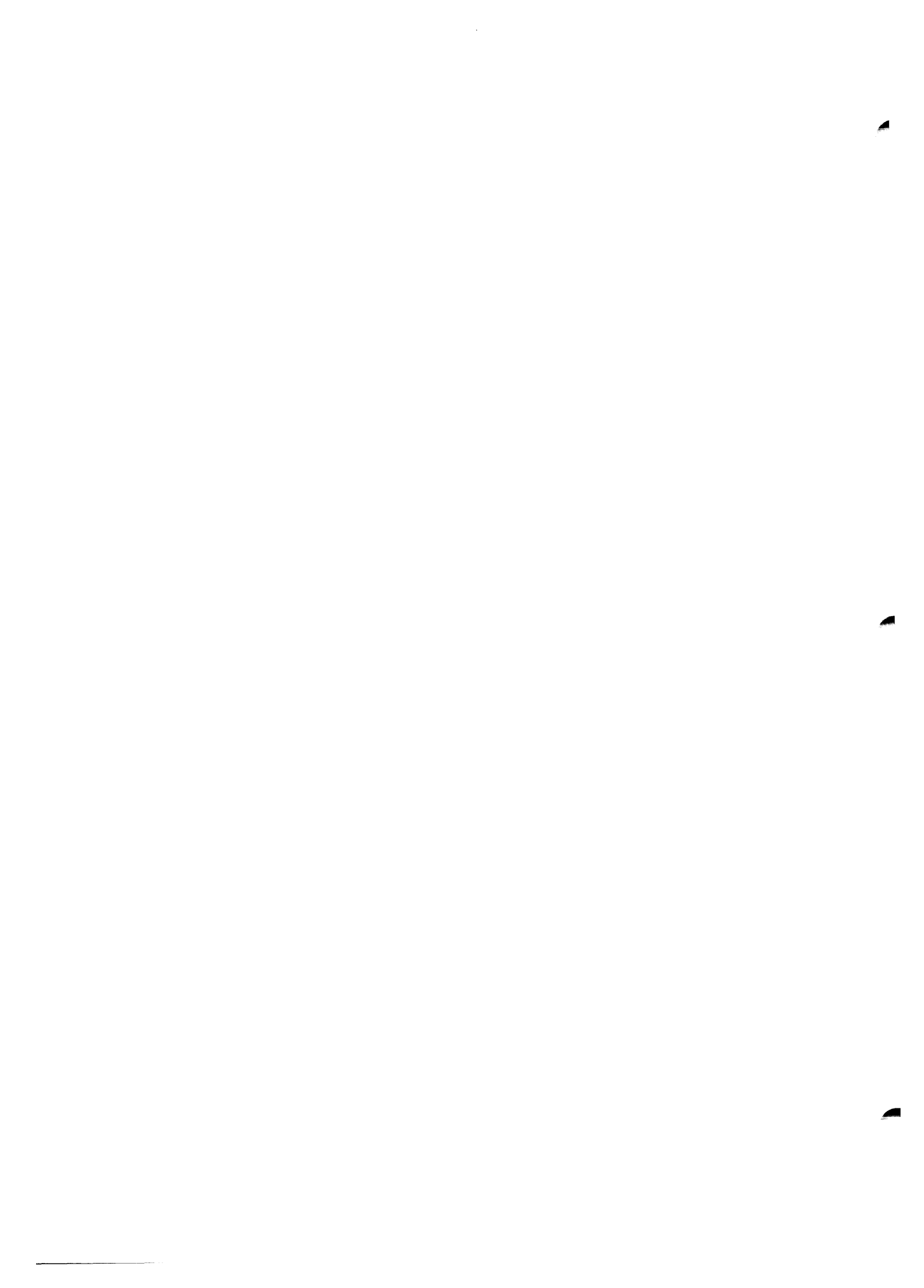
Illustrations

Figure	Title	Page
1-1	Physical Layout of DNOS	1-2
1-2	990 TILINE Logical Address Space Example	1-4
1-3	DNOS Task Structure	1-6
1-4	DNOS Queue Structure	1-7
1-5	DNOS System Flow	1-10
2-1	Volume Information Format (2 Sheets)	2-11
2-2	Partial Bit Map	2-15
2-3	Directory Structure	2-16
2-4	Directory File Structure	2-17
2-5	Directory Overhead Record Format	2-18
2-6	FDR Format (2 Sheets)	2-19
2-7	ADR Format	2-23
2-8	CDR Format	2-26
2-9	KDR Format	2-31
2-10	Dump of a Directory File	2-33
2-11	Sequential File Format (2 Sheets)	2-35
2-12	Blank-Suppressed Record	2-38
2-13	KIF Structure	2-43
2-14	Sequential Record Placement Method	2-44
2-15	B-Tree Block Format	2-45
2-16	B-Tree Example (2 Sheets)	2-48
2-17	Data Block Format	2-50
3-1	SCI Modes of Operation	3-2
3-2	SCI Access to Logical Names and Synonyms	3-2
4-1	Format of RPUDAT Module	4-3
4-2	User-Defined SVC	4-5
5-1	Interrupt Decoder for Single Device	5-5
5-2	Interrupt Decoder for Multiple Devices	5-6
5-3	Expansion Chassis Interrupt Decoder (3 Sheets)	5-7
5-4	Asynchronous Multiplexer Interrupt Decoder (2 Sheets)	5-11
5-5	Interrupt Decoder Return Routine	5-13
5-6	Illegal Interrupt Routine	5-13
5-7	DSR Structure	5-14
5-8	Memory Map for DSR Execution	5-15
5-9	Hardware Interrupt Processing	5-17
5-10	Asynchronous DSR Structure	5-33
5-11	Asynchronous DSR Logic Flow	5-35
5-12	Interrupt Processing Flow	5-38
5-13	Asynchronous Data Structure Linkages	5-40
5-14	DSR Listing Example (16 Sheets)	5-53
7-1	Access Groups and Secured Files	7-5
7-2	Creating an Access Group	7-6

Figure	Title	Page
8-1	System Loader Error	8-2
8-2	General Information Block	8-7
8-3	Task States and JSB List (2 Sheets)	8-9
8-4	Bit Assignment for the Flag Word	8-11
9-1	Functions of Message File Utilities	9-10

Tables

Table	Title	Page
2-1	Format Information for Supported Disks	2-9
3-1	Special SCI Characters	3-5
3-2	SCI-Maintained Synonyms	3-10
3-3	Message Processing Synonyms	3-11
3-4	Valid Field Prompt Types	3-12
3-5	SCI Primitive Notation	3-17
3-6	SCI Primitives	3-17
3-7	Command Privilege Levels	3-19
3-8	Disallowed SVCs for .SVC Primitive	3-43
4-1	Request Definition Block (RDB) Format	4-2
4-2	RIB Format	4-4
5-1	DSR/TSR Entry Points	5-16
5-2	Asynchronous Device Support	5-31
5-3	HSR Object Modules	5-39
5-4	HSR Baud Rate Codes	5-47
5-5	Controller Type Codes	5-49
8-1	System Loader Phases	8-3
8-2	System Loader (Flashing) Crash Codes	8-3
8-3	XANAL Commands	8-5
10-1	ASCII Codes for Special Language Characters	10-3
10-2	JISCII Codes for Japanese Characters	10-4
10-3	Collating Sequences for All Supported Languages	10-6
11-1	DX10/DNOS System File Names	11-4



DNOS System Overview

1.1 DNOS SYSTEM STRUCTURE

The Texas Instruments Distributed Network Operating System (DNOS) is a general-purpose operating system for the Texas Instruments Business Systems and Model 990 Computers. It allocates system resources to the jobs executing on the computer system to provide maximum performance based on information supplied by the user and on real-time performance analysis. In the DNOS environment, a job is a set of one or more cooperating programs (tasks) and performs one or more functions.

DNOS has two parts; one is memory resident and the other is disk resident. The memory-resident portion, called the *kernel*, is loaded into memory during the initial program load (IPL). (The memory-resident parts of the system are linked together when the operating system is generated.) The kernel must reside in main memory before any processing can occur.

The kernel provides support for activating and terminating tasks and for processing supervisor calls (SVCs) and interrupts. It also provides support for system tasks in the form of common routines and functions callable from system tasks.

Disk-resident modules of the system are brought into main memory from disk storage as they are needed to perform specific functions. This disk-resident part of DNOS consists of various system tasks that perform primarily queue-serving functions.

Figure 1-1 shows a physical layout of DNOS. In the figure and elsewhere in this manual, the right angle bracket (>) preceding a value indicates a hexadecimal value.

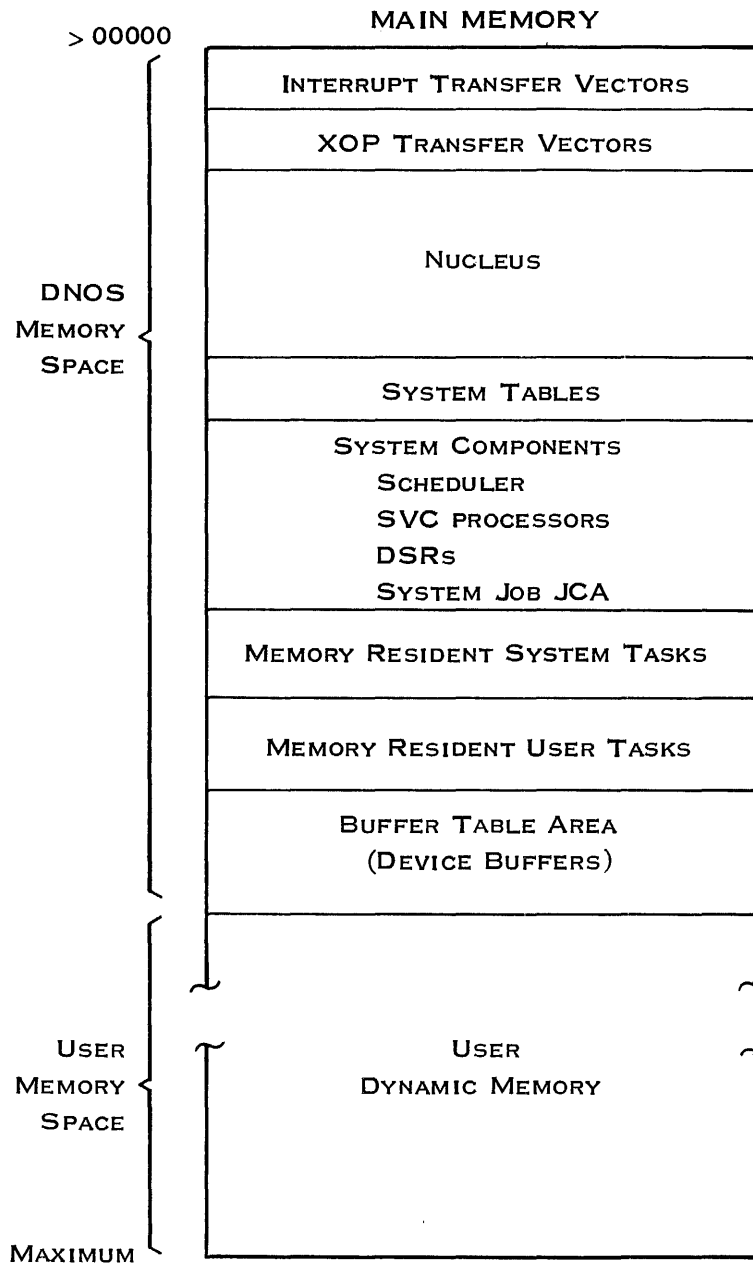
The kernel includes the following parts:

Transfer Vectors

A transfer vector is a pair of consecutive memory words. The first word contains the address of a 16-word workspace register area. The second word contains the address of a subroutine entry point. A transfer vector is used to perform a type of control transfer called a *context switch*. An example of a transfer control switch is the transfer of control to an interrupt subroutine when an interrupt occurs. Business Systems and Model 990 Computers support 16 transfer vectors for interrupts. They also support 16 transfer vectors for extended operations (XOPs). XOPs perform system-defined functions implemented by software.

Nucleus

The nucleus contains routines that support queuing, synchronization, linking to other routines, and managing the scheduler. The nucleus portion also includes the system table area.



2279382

Figure 1-1. Physical Layout of DNOS

System Tables

All the data structures needed to support system operations are located in the table areas. These areas include segment management table areas, file management table areas, the system job communication area, and the partial bit map for the system disk.

System Components

System components include the scheduler, Supervisor Call (SVC) processors, and the device service routines (DSRs). You use these components temporarily, as segments mapped into the logical memory space as required.

In addition to the kernel, DNOS memory space also contains the following:

System Tasks

The memory-resident system tasks are loaded into memory next to the system segments. The memory-resident tasks include file management, disk management, and the task loader.

Memory-Resident User Tasks

All user-written, memory-resident tasks reside in the area next to the memory-resident system task area.

Partial Bit Maps

The partial bit maps for all disks except the system disk reside below the memory resident tasks.

Buffer Table Area

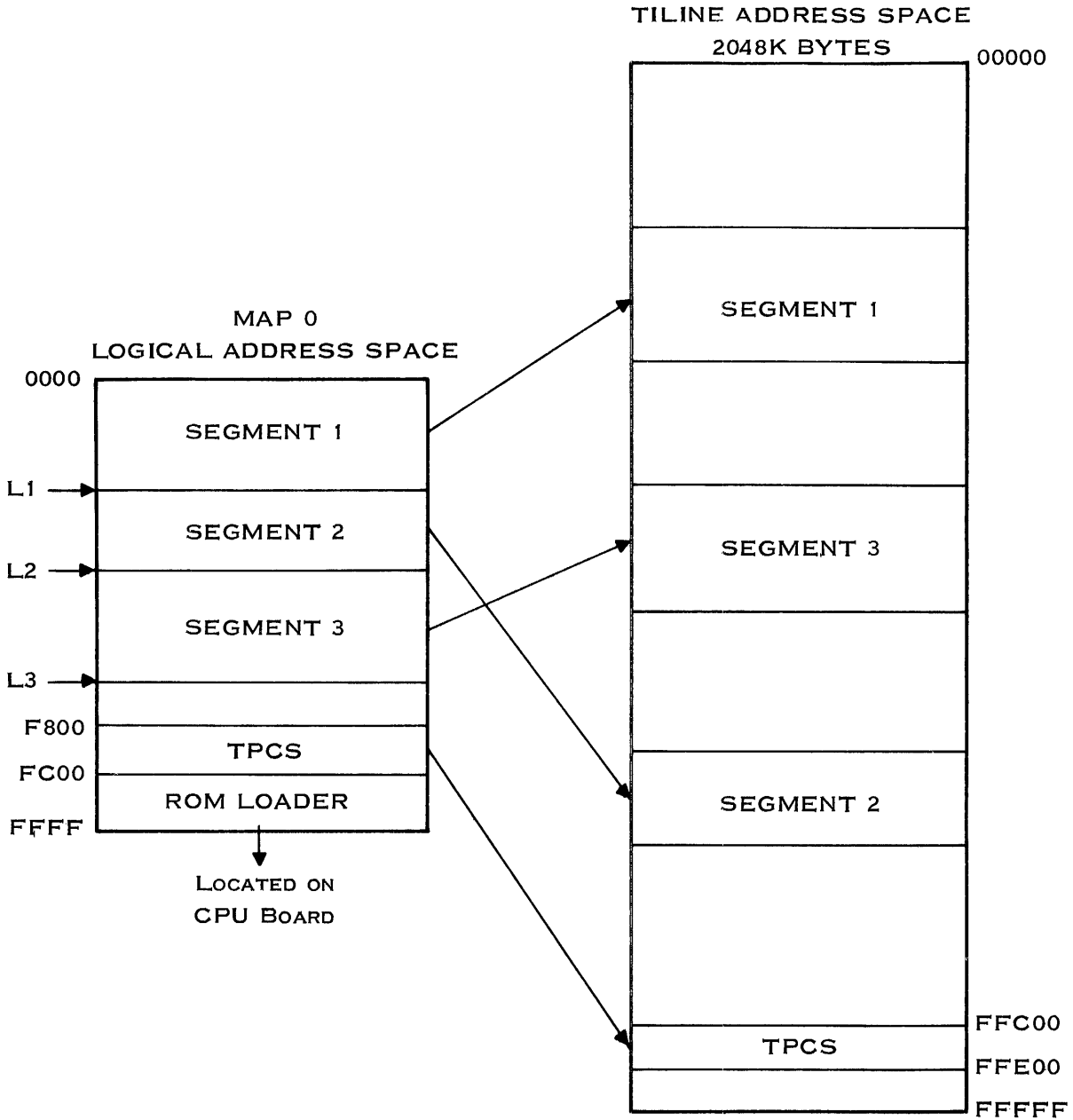
The buffer table area (BTA) is allocated immediately after the memory-resident user tasks. This expandable area includes buffers used for handling input and output for devices.

The rest of memory is available for user tasks, language processors, and utilities. The user dynamic memory is allocated to tasks from the end of physical memory to allow the buffer area to expand.

1.1.1 System Memory Mapping

The hardware map option provides three map files; each defines up to 64K (K equals 1024) bytes of logical memory space. This space can be located anywhere in the 2048K-byte TILINE address space and can be divided into as many as three segments.

Each map file provides three 16-bit bias registers (B1, B2, and B3) and three limit registers (L1, L2, and L3). The limit registers contain values that define the lengths of the three segments. These segments form the 64K bytes of logical address space. The bias registers contain values that define the physical locations of the three segments. Refer to Figure 1-2 for a view of the logical and TILINE address space with map option. The logical addresses in the three segments are mapped into TILINE beet (32-byte) addresses.



2279383

Figure 1-2. 990 TILINE Logical Address Space Example

Usually, the operating system dedicates the three map files to software functions. The system uses map 0 for operating system code and for facilitating special hardware features. Transfer vectors for interrupts and XOPs are in map 0. This map is one means of addressing the TILINE™ peripheral control space and read-only memory (ROM) loader. The TILINE peripheral control space is a range of TILINE addresses reserved for access to TILINE device controllers. After setting up a map file, a user might also access TILINE peripheral control space using long-distance instructions. The ROM loader is a program that executes during IPL to load the operating system into memory.

Both system tasks and user tasks use map 1. The long-distance instructions use map 2 to access memory outside the segments currently mapped into the logical memory space.

1.1.2 Fundamental Structure of DNOS

The fundamental structure of DNOS is the job structure. A job is a collection of cooperating tasks (programs) that you initiate or a program automatically initiates to perform one or more functions.

The task structure is within the job structure. Many of the DNOS system tasks are queue servers, tasks dedicated to processing entries on queues. Queues and queue servers play an important role in maintaining system flow. For storing system data, DNOS maintains a set of system files.

1.1.2.1 Job Structure. A job is the fundamental entity that receives logical resources. DNOS maintains a job status block (JSB) for each job in the system. The JCA is a block of memory that contains control information for a specific job including the job's priority, name, ID, and related information. The JSBs are created in the system table area as a linked list for the jobs in the system.

The JSB provides a link to the job communication area (JCA), which contains a queue of the tasks in the job, the semaphores defined for the job, and their job-local information. Semaphores provide synchronization of tasks within a job. The JCA for the job currently executing is kept in memory.

1.1.2.2 Tasks. A task is a program that executes under control of the operating system. It consists of an address space, a program counter, a workspace pointer, and a status register. At least one task exists for each job.

Associated with each task is a task status block (TSB), which is a block of memory containing information about the task. The TSBs within each job are on a task queue located in a JCA. Each TSB can be placed on the active queue, the waiting-on-memory (WOM) queue, or the waiting-for-table-area queue, according to the priority order within the job.

Figure 1-3 shows the DNOS task structure.

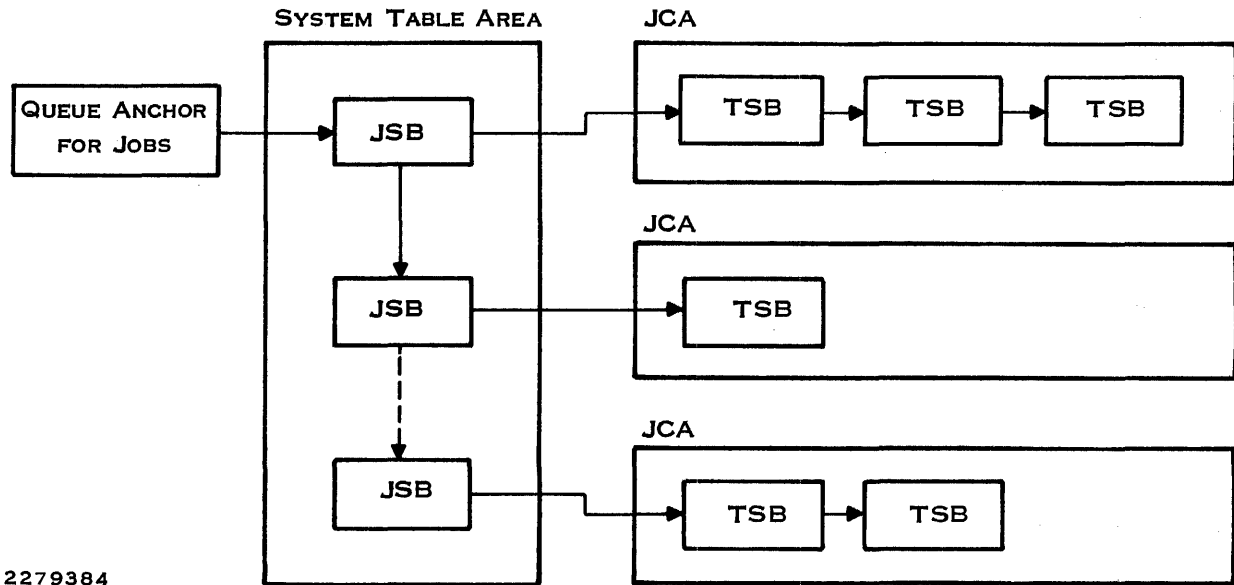


Figure 1-3. DNOS Task Structure

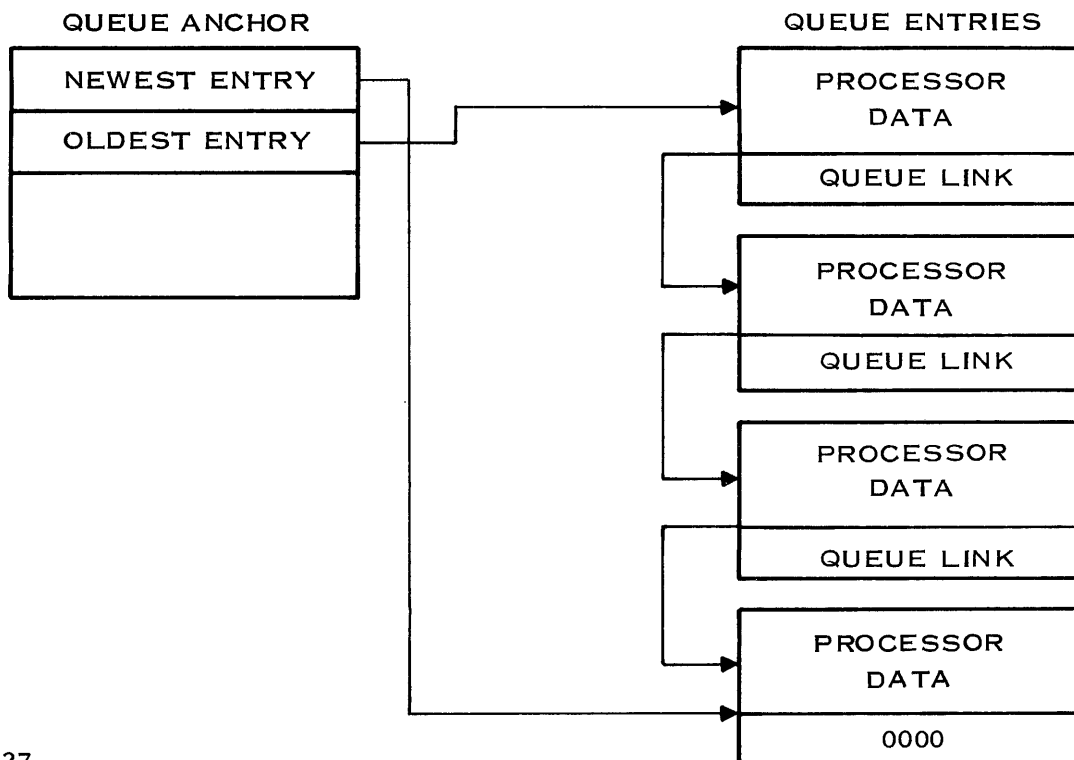
1.1.2.3 Queue Servers. Information queues and queue servers are important concepts in DNOS. A queue is a first-in, first-out list of data to be processed. In DNOS, each queue consists of a queue anchor and the queued blocks of data in memory. The queue anchors are located in the memory-resident nucleus and in the JCAs. Each queue anchor contains the address of the newest and the oldest queue entries. Each data item or block is linked to the next data item or block in the queue.

A queue server is a task that processes its associated queue. Placing an entry in the queue activates the queue server. Operating as a system task under DNOS, the queue server dequeues an entry and then processes it. The queue server continues to dequeue and process queue entries until the queue is empty, at which time the server suspends itself or terminates.

Figure 1-4 shows the DNOS queue structure.

1.1.2.4 System Files. Various functions of DNOS use system directories and files. Some are required if you choose certain options; otherwise, you can remove them. Also, other software systems and the languages used with DNOS place S\$ files on the system disk. Remove these only if you are sure your environment does not need them.

The system disk is the disk from which IPL is done. It is known by the device name set during system generation. For example, if IPL is done from DS04, the file .S\$CMDS is the same as DS04.S\$CMDS.



2284927

Figure 1-4. DNOS Queue Structure

DNOS functions use the following directories:

Directories	Description
.\$CDT	Files used by the system to process keyboard bids at devices.
.\$CMD	Command procedures provided with DNOS to access the utilities using a System Command Interpreter (SCI) environment.
.\$EXPMSG	Key indexed files (KIFs) of expanded error and status messages. If you remove this directory (your option), you cannot use the question mark (?) key to display details on the screen when a message appears.
.\$MSG	Relative record files of basic error and status messages. If you remove this directory (your option), all messages appear in abbreviated form, as in the following: SVC-INTERNAL CODE > 0027 .PRINT.OUT.

.S\$SDTQUE	Files of spooler data, one for each generated system. Do not delete these unless you deleted the generated system or you need to recreate one of the files.
.S\$SGU\$	Files created by system generation. You can delete subdirectories for systems no longer in use, but you should keep those for systems in use. For further information about these files and directories, refer to the <i>DNOS System Generation Reference Manual</i> .
.S\$SYSLIB	Overlay management for automatic overlay loading.
.S\$SYSTEM	Special systems programming command procedures and the software configuration history file. Section 12 describes these procedures.
.S\$USER	Directory with subdirectories for each user ID defined for the system. Each subdirectory contains user synonyms and logical names; you must not delete any of these subdirectories. The SYSTEM and SYSMGR user IDs are created during IPL.
.SCI990	Linkable object for the SCI interface (S\$) routines.

DNOS functions use the following files:

Files	Description
.S\$ACT1, .S\$ACT2	Accounting log files. You need these if the accounting option is enabled. You cannot modify the file currently in use by the system.
.S\$CLF	Capabilities list file used by SCI. Used in conjunction with the S\$USER directory.
.S\$CRASH	Crash file to which a system crash can be written.
.S\$DIAG	File used by online diagnostics when checking the state of the disk.
.S\$IPL	System loader.
.S\$ISBTCH	Initial batch stream executed during the initialization of the system after an IPL.
.S\$ISLIST	Listing file for .S\$ISBTCH. You can delete this file if you do not need the listing.
.S\$LANG	Languages program file.

.S\$LOG1, .S\$LOG2	System log files used to record error and status information about the active system. You cannot modify the file currently in use by the system.
.S\$MVI	File used by the Modify Volume Information (MVI) processor to record changes to the disk.
.S\$PWCS	Image file for performance tools microcode. This file is present if the DNOS performance package is installed.
.S\$ROLLD.S\$ROLLA	System roll file used for swapping segments from memory.
.S\$SCA	File of information about users that is used by the log-on task and SCI.
.S\$SECURE	Program file containing support programs for file security.
.S\$SHARED	Shared program file, used for sharable procedures and special tasks provided by the system. This program file reserves task IDs and procedure IDs >00 through >2F for software provided by Texas Instruments. All other IDs are intended for users.
.S\$SHIP	Kernel program file for the system shipped to the users. You can delete this file if you use a different system as the standard system.
.S\$UTIL	System utilities program file.

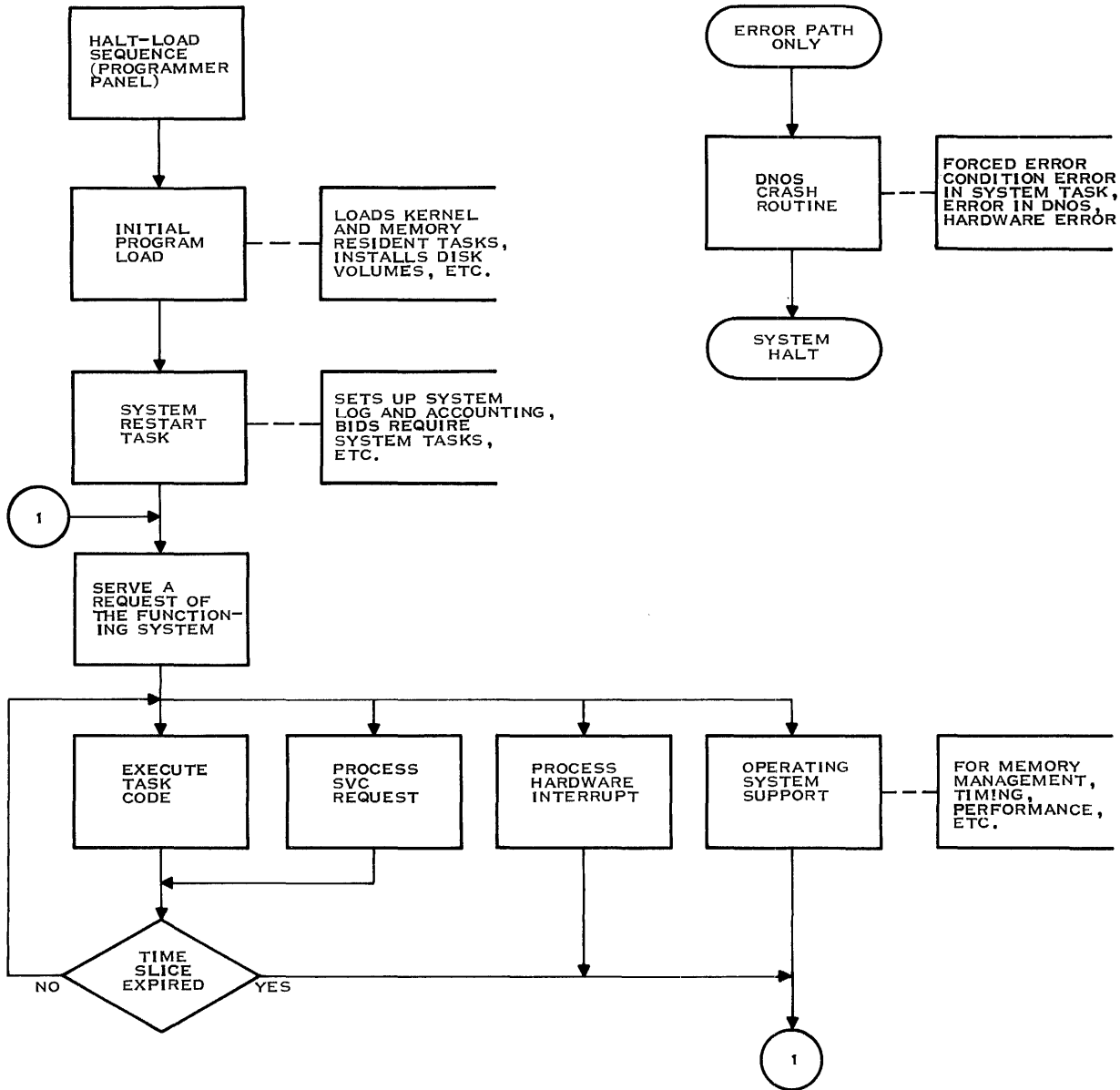
1.2 CHANNELS FOR DNOS FUNCTIONS

DNOS functions use the following channels. Each channel is served by an owner task in the S\$UTIL program file.

Files	Description
.S\$ACCCHN	Channel that processes accounting file entries.
.S\$DSTCHN	Channel used by the spooler device scheduler.
.S\$MAIL	Channel used for the Create Message (CM) function and other SCI message functions.
.S\$OPER	Channel used for system operator functions.
.S\$SPOOL	Channel used by the spooler.
.S\$XBJ	Channel used for processing the Execute Job (XJ) and Execute Batch Job (XBJ) commands.

1.3 SYSTEM FLOW

The following paragraphs describe how the operating system responds to requests for service. Figure 1-5 shows the flow of control and information in DNOS.



2279386

Figure 1-5. DNOS System Flow

1.3.1 Initial Program Load (IPL)

System operation begins with IPL. The IPL program is loaded into main memory and performs housekeeping functions (such as determining the size of physical memory and initializing physical memory). The IPL program then relocates into upper memory and reads the kernel into lower memory. Next, the IPL program performs a variety of system initialization functions. The operating system is then activated.

1.3.2 System Idle State

DNOS enters the idle state immediately after IPL and when no programs or users require system services. However, the logical structure of the system changes dynamically when tasks are executed under control of the operating system.

1.3.3 Task Activation

Bidding a task is the process of preparing a task for execution. This process involves building and initializing the necessary data structures and activating the task. If the task procedural segments are already in memory, the system checks to see that the task is not being killed and that its job is not terminating. If these conditions are met, the task is placed on the active queue. Otherwise, task activation aborts.

If the task procedural segments are not in memory, the task is put on the waiting-on-memory (WOM) queue to be processed by the task loader. After the task is loaded into memory, the checks described for tasks already in memory are performed. Either the task is placed on the active queue or task activation aborts, as appropriate.

1.3.4 Task Scheduling

The task scheduler places tasks into execution. First it selects a task to execute; then, it instructs the central processing unit (CPU) to begin executing the task. The task executes until it releases control of the CPU. Then, the scheduler selects the next task for execution.

Each JCA contains a queue of TSBs for tasks ready to execute in priority order. Each JSB carries the priority of the highest-priority task on its active queue; the queue of active JSBs is ordered by this priority.

The scheduler selects the highest-priority active task for execution. When a task reaches the end of its allotted execution time, its TSB goes back to the JCA active queue if the task is to remain active; the task remains unqueued if it is to be suspended. When a task suspends, the scheduler might need to change the priority of the highest active task in the JSB and reorder the JSB on the JSB queue.

1.3.5 Time Slicing

Time slicing is the technique of executing each task in turn for a specified period of time. The clock interrupt processor controls timing. The clock interrupt routine counts the number of clock cycles during task execution. When the count reaches a specified number, the routine reschedules the task. Each clock tick is either 8.33 milliseconds (for 60-Hertz line frequency) or 10 milliseconds (for 50-Hertz line frequency).

1.3.6 XOP Processing

When the scheduler places a user task into execution, the task controls the CPU until its time slice ends. The only exceptions are when an external interrupt occurs or a task issues an XOP instruction. The I/O subsystem activates immediately in response to a device interrupt. A context switch occurs when an XOP instruction is issued.

When an XOP 15 is issued, control passes via the XOP transfer vector table to the SVC decoding routine; this routine is the XOP processor for XOP 15. It determines which SVC is desired by decoding the SVC code in the call block and then relinquishes control to the SVC processor.

If a queue server processes the request, the SVC preprocessor buffers the call block into the system table area and queues the buffered call block onto the proper queue. This activates the associated queue server task and suspends the task that issued the SVC. If system code that is not a queue server processes the request, the SVC is completed and control returns immediately to the task that issued the SVC.

1.3.7 Execution Priorities

The scheduling of DNOS tasks is based on run-time priority. Run-time priorities have a range of 0 (high) through 255 (low). To calculate a run-time priority when a task is bid, DNOS must first look at the installation priority of the task. An installation priority is assigned to a task when it is installed in a program file. When assigning run-time priorities, DNOS differentiates between tasks that are either priority 0 or real-time tasks and those that are not.

The run-time priority of real-time tasks and priority 0 tasks is identical to their installation priorities.

The run-time priority of all other tasks is influenced by the following three factors:

- The installation priority of the task (1, 2, 3, 4)
- The mode of the task (whether foreground or background)
- The priority of the job under which the task is running

Be careful to note that the installation priority given a task (1, 2, 3, 4) that is not a priority 0 or real-time task is relative. Such tasks normally have run-time priorities between 128 and 255.

The run-time priority of a real-time or priority 0 task is the same as its installed priority. Therefore, a system task with an installed priority of 0 has a run-time priority of 0. A real-time task with an installed priority of 56 has a run-time priority of 56.

System tasks usually have an installed priority of 0. Real-time tasks have an installed priority range between 1 (high) and 127 (low). Therefore, run-time priorities are also in the same range.

The run-time priority of all other tasks is handled in a different manner. Tasks in this group usually pick up a run-time priority between 128 and 255 (refer to the paragraph describing dynamic modification of run-time parameters for exceptions).

First, DNOS looks at the installed priority (1, 2, 3, or 4) of the job in which the task is being bid and whether the task is executing in foreground or background. When determining which installation priority to give a task, you should normally use the following scheme:

- Assign priority 1 to foreground tasks that are heavily interactive.
- Assign priority 2 to foreground tasks that are compute-bound.
- Assign priority 3 to only those tasks that always execute in background, as priority 3 is the lowest of the four priority classes.
- Assign priority 4 to tasks that alternate between an I/O bound state and a compute-bound state. This priority level is proper for most tasks in a computing environment. The run-time priority of a priority 4 task running in foreground is lower than that of a priority 1 running in foreground task but greater than that of a priority 2 task running in foreground.

Second, DNOS looks at whether the task is executed in foreground or background. Any task, regardless of its installed priority, is treated as a priority 3 task if you bid it in background mode.

Third, DNOS is heavily influenced by the priority of the job in which the task is bid (range: 0 (high) through 31 (low)). A low-priority task in a high-priority job often has a higher run-time priority than high-priority tasks in a low-priority job. For example, a background task in a priority 2 job is assigned a higher run-time priority than a foreground task in a priority 29 job.

1.3.8 Dynamic Modification of Run-Time Parameters

DNOS allows the dynamic modification of run-time priorities. This is called the priority modification option. If you enable this option, the run-time priority of a task varies during its execution.

To enable this option, use the Modify Scheduler/Swap Parameters (MSP) SCI command to modify dynamic priority range parameters. There is one parameter for each of the four installed task priorities (1, 2, 3, or 4). Unless you are very familiar with dynamic priority range parameters, you should use the following values when you modify the parameters: 12, 12, 12, 12. These values yield the maximum system performance for most environments. To disable this option, set the parameters back to their default values (0, 0, 0, 0).

If you accept the defaults (0, 0, 0, 0) for the dynamic priority range parameters (refer to the Modify/Swap Parameters SCI command), tasks in this group pick up a run-time priority between 128 and 255. Modifications to the parameters, however, can yield a run-time priority higher than 128.

To understand this process, consider the following. As a task executes, an indicator shows whether the task is I/O-bound or compute-bound. DNOS uses this indicator to modify the run-time priority of tasks (raising priority for I/O-bound tasks and lowering it for compute-bound tasks).

The degree to which a run-time priority can vary for tasks depends on the value of the dynamic priority range for that priority class. For example, if you use the MSP command to assign dynamic priority range parameters of 8, 12, 0, and 32, the following events occur:

- The run-time priority of priority 1 and priority 2 tasks would vary by plus 12 (I/O bound tasks), since 12 is the smallest allowed variation.
- The run-time priority of priority 3 would be unaffected.
- The run-time priority of priority 4 tasks would vary by plus 24.

Test results show that modifying dynamic priority parameters can improve the mean response time of DNOS in some environments.

An aging factor can further modify the run-time priority. The priority of an older task is raised slightly more than the priority of a newer task. To raise the priority, the power of 4 that represents the execution time in seconds is used. That is, a task that has executed for 4 seconds is raised 1 priority level; one that has executed for 16 seconds is raised 2 levels, and so on. At the end of 18 hours of execution, the priority of a task is raised 8 levels.

1.3.9 Task Termination

A task terminates when one of the following occurs:

- The task issues a termination SVC.
- Another task issues a Kill Task SVC for this task.
- The task aborts by executing an illegal operation.

If the task specifies end action (execution after abnormal termination), execution resumes at the specified end-action address for a certain length of time. Otherwise, the task releases its resources and goes to the terminate task queue, where the termination processor task deallocates it.

1.3.10 Clock Interrupt Processor

The clock interrupt processor gathers performance statistics, keeps track of time, and decides when a system time unit has expired for the executing task. The time and date appear in the following form: year, day (Julian), hour, minute, second, and tick (8.33-millisecond unit). A 32-bit tick counter also keeps track of time in clock ticks. The time, date, and 32-bit tick counter are updated on each clock tick.

1.3.11 Internal Interrupt Processor

Instruction execution errors (for example, illegal opcodes and privileged instructions) cause internal interrupts (interrupt level 2). The internal interrupt processor handles these interrupts. If an interrupt occurs in task code, the processor kills the task or puts it into its end-action code; control returns to the scheduler. However, if the error occurs in operating system code, in interrupt processing code, or while scheduling is inhibited, the processor calls the system crash routine.

1.3.12 System Crash Routine

When a module detects an internal operating system error, it branches to the system crash routine and passes a crash code that indicates the type of error. The crash routine halts the system and displays the crash code on the programmer panel of the computer. Pressing HALT and RUN on the programmer panel saves the state of the system at the time of the crash and writes an image of memory to the crash file on disk. You can then analyze the crash.

1.4 IPL AND SYSTEM LOADERS

IPL is the process of loading the operating system into memory. Before you can enter any system command into the system for execution, the IPL procedure must bring DNOS into memory. To perform an IPL, press in sequence HALT and LOAD on the front panel of the computer. For a Business System 300, turn the power off and on to perform an IPL.

When an IPL procedure completes, the system restart task is bid. The task performs the following initialization activities:

- Defining channels needed by DNOS
- Assigning system-required global logical names
- Creating log and accounting files
- Deleting temporary files
- Creating the SYSTEM user ID

The task also performs initialization activities that enable DNOS to offer the security option:

- It creates the SYSMGR user ID.
- It creates the SYSMGR access group. (The SYSMGR access group is created only when the S\$CLF file needs to be created.)

The IPL process checks to see that the crash file on the system disk is large enough to contain the entire system memory image. If the file is not large enough, the IPL process attempts to delete the existing .S\$CRASH file and creates a larger one. If the IPL process encounters an error when it tries to delete the file, the first person to log on the system is shown a message that says that the crash file is either too small or it does not exist.

The file .S\$ISBTCH serves as a batch stream for adding unique system procedures that are performed immediately following IPL. For example, you can include procedures to assign global logical names, initialize certain functions, monitor devices, or delete certain directories. Executing .S\$ISBTCH also invokes the SCI command procedure M\$00. This batch stream executes in a job named SY\$INT. For this job, the synonym \$\$UI is not defined.

The following paragraphs describe what happens between pressing the LOAD switch and initiating a job from a terminal.

Three loaders are involved in an IPL procedure: the ROM loader, the program image loader, and the system loader. The ROM loader brings the program image loader into memory from the system disk. The program image loader locates the system loader on the disk and loads the system loader. The system loader loads the system image and transfers control to the system. The following paragraphs discuss each loader in detail.

1.4.1 ROM Loader

The ROM loader (bootstrap loader) resides in TILINE peripheral control space starting at location > FC00. You can program this loader to load from devices accessible on the TILINE bus and on the communications register unit (CRU). The IPL is performed from a system disk (a TILINE device). (This manual does not describe using the ROM loader for other devices.)

Location > 80 contains a negative value that indicates the TILINE device to be used as the load device. Location > 82 contains the TILINE address of the load device. This address specifies the location of the TILINE peripheral control space for loading the TILINE device commands. The default is > F800. To load the system on a 990/10 or 990/12 computer (using a disk controller) at an address other than > F800, you must change the contents at location > 82.

Refer to the *ROM Loader User's Guide* for a description of how to use the ROM loader and how to modify the value for the default load device.

1.4.2 Program Image Loader

The program image loader resides on track 1 of every DNOS-formatted disk that was specified as a system disk. It can load any stand-alone program from an image file or object file on disk into memory. The following criteria determine the program to be loaded:

- If the diagnostic flag in the volume information is nonzero, the diagnostic task is loaded and the flag is reset to zero. Section 2 describes the volume information in detail.
- If no diagnostic is specified, the loader checks to see if the file pathname of either a primary or a secondary system loader is specified. If so, the image loader loads the system loader indicated by the flag, as follows:

- 0 — Load primary
- 1 — Load secondary
- 1 — Load secondary

If the flag equals -1, the image loader resets the flag to 0.

- If no system loader is specified, the image loader loads the system image indicated by the image flag, which is used in the same manner as the system loader flag.

The program image loader normally loads a program image starting at memory location > A0. This default load address is stored in the second word of the loader. You can change it by using the Modify Absolute Disk (MAD) command.

1.4.3 System Loader

The system loader resides on the system disk in an image file called S\$IPL. The program image loader loads it into memory. The system loader executes with interrupts masked to level 2, inhibiting interrupts from devices. Once loaded into memory, the loader initializes physical memory and determines the actual size of physical memory in the computer.

As the loader finishes a particular phase of the load process, it displays the phase on the programmer panel indicators, starting at the leftmost indicator.

The following lists in sequential order the phases of the load process:

Phase Number	Description
1	Loader relocation complete
2	Successful opening of kernel program file
3	Successful loading of root, verification of system version, and loading of writable control source (WCS)
4	Successful loading of special table areas
5	Successful initialization of system overlay table and crash file
6	Successful loading of JCA segments
7	Successful loading of DSRs and scheduler
8	Successful loading of memory-resident system tasks
9	Successful loading of user memory-resident tasks

The loader first relocates itself into high-order physical memory. It then opens the program file that contains the kernel and loads the WCS (when applicable). Next, the loader loads the system root and initializes the crash file. If a crash occurs during the remainder of the load operation, the crash file contains useful information about the crash. Special table areas are loaded next, followed by the system job JCA, DSRs, the scheduler, and memory-resident system tasks.

Next, the loader performs some special initialization, such as the following:

- Determining which of the disk drives defined is the disk from which the system was loaded and marking it as the system disk
- Installing the system disk
- Creating file descriptor blocks (FDBs) for the system files used during the load process

The next step in the load process is to install all online disk volumes. The final phase of the loader execution is allocation of the buffer table area (located in user memory, immediately following the memory-resident part of the operating system and all memory-resident tasks) and initialization of the system anchors for the free user memory of the buffer table area. The memory containing the loader is considered part of user memory. After all initialization is performed, the loader transfers control to the power-up interrupt processor of the operating system.

1.5 DNOS SUBSYSTEMS

DNOS includes several subsystems that implement capabilities including job management, segment management, name management, and interprocess communication (IPC). User commands interpreted by SCI can access these capabilities. User programs access these capabilities by executing SVCs. The following paragraphs describe these subsystems and the relevant commands.

1.5.1 Job Management

A job is an entity that performs a user-defined function within the system. It can include one or more tasks, and it can be either interactive or batch.

An interactive job is initiated when a user logs on at a terminal. You can also initiate an interactive job by entering an Execute Job (XJ) command. The terminal specified in response to the STATION ID prompt is the terminal used for interaction.

A batch job consists of one or more tasks that do not require interaction with a terminal and that execute in the background. The commands that direct the execution of the task or tasks of the job are supplied in a file. To initiate a batch job, enter an Execute Batch Job (XBJ) command.

Job management is the subsystem of DNOS that performs the system functions required to initiate, execute, and terminate jobs. Management of resource allocation by jobs provides a level of protection between jobs. Once a job is initiated, execution of the tasks of a job is independent of the execution of tasks within other jobs.

1.5.1.1 Supervisor Call (SVC). The Job Management Request SVC is the interface of a task with the job management subsystem. This SVC performs the following job management operations:

- Create Job
- Halt Job
- Resume Job
- Modify Job Priority
- Map Job Name
- Kill Job
- Get Job Information

A task requests creation of a job by requesting a Create Job operation. If the request is valid, the job is created with a unique job ID. This ID is used throughout the system as the identifier of the new job. The job is set into execution if the job limit is not exceeded. If the limit is exceeded, the job is queued, waiting for an executing job to terminate.

Various control capabilities are available for interpreting and modifying the status of jobs. The use of these capabilities depends on the requester's ability to prove ownership (by user ID) of the job. The types of requests available include those to show job status, kill the job, halt the job, resume the job, and modify job priority.

1.5.1.2 SCI Commands. The following commands are available through SCI to execute jobs and to perform various operations on jobs currently executing under the user's ID:

Execute Batch Job (XBJ)

The XBJ command creates a job having batch SCI as its initial task.

Execute Job (XJ)

The XJ command creates an interactive job with operating parameters differing from those of the creating job.

Show Job Status (SJS)

The SJS command displays the status of any jobs currently executing under the user's ID.

Halt Job (HJ)

The HJ command suspends a job currently executing under the user's ID.

Kill Job (KJ)

The KJ command forces termination of a job currently executing under the user's ID.

Resume Job (RJ)

The RJ command resumes execution of a job that has been previously halted.

Modify Job Priority (MJP)

The MJP command modifies the priority of a job while it is executing.

List Jobs (LJ)

The LJ command allows you to list the current status of a particular job or all jobs on the system.

1.5.2 Segment Management

The segment management feature enables tasks to dynamically change the current segment set mapped by the task. This feature also enables a task to guarantee access to a segment until it is no longer needed and enables a task to write segments to disk.

Segmentation allows more than three segments of a task to be in memory. The program can map three segments in the program address space simultaneously. This feature allows faster execution speed than overlay loading. If enough memory is available, it also enables a program to exceed the 64K-byte address boundary.

The two types of segments are memory-based and disk-based. The Create Segment operation of the Segment Management SVC creates a memory-based segment at run time. The newly created segment is mapped into one of three segments by specifying the segment position or by replacing an existing spare one using the segment run-time ID. Segments not in a task address space are released from memory if the segment reserve count is 0. The reserve count keeps track of the current usage of the segment.

The Reserve Segment operation of the Segment Management SVC increments the reserve count. For a task to retain access to segments not in the task address space, the user must request a Reserve Segment operation. Such segments are reserved until the user requests the Release Segment operation of the SVC.

A disk-based segment is either a segment installed in a program file or a physical record of a relative record file. To install a segment, use the Install Task (IT), Install Procedure (IP), or Install Program Segment (IPS) command. One, two, or three segments can be brought into memory when the task is loaded for execution. This is accomplished by the IT command with specification of attached procedures. The Change Segment operation of the Segment Management SVC loads installed segments into memory while the program is executing. The Force Write Segment operation can write the disk-based segments that are updateable back to their home file position.

You can place an unmapped segment physically in memory by requesting a Load Segment operation. The Unload Segment operation releases the segment from memory. The Set Exclusive Use of Segment operation prevents other tasks from accessing an unmapped segment. The Reset Exclusive Use of Segment operation releases the segment from exclusive use. Like a Reserve Segment operation, the Exclusive Use of Segment operation allows a task to retain access to a segment not in the task address space.

The Segment Management SVC supports the following operations:

- Change Segment
- Create Segment
- Reserve Segment
- Release Segment
- Check Segment Status
- Force Write Segment
- Set/Reset Release/Modifiable Flag
- Load Segment
- Unload Segment
- Set Exclusive Use of Segment
- Reset Exclusive Use of Segment

1.5.3 Name Management

The following paragraphs introduce the concept of logical names and describe their uses with files.

1.5.3.1 Logical Names. A logical name is a system variable from one to eight characters long, defined by the user. The value of the logical name is a pathname or device name. The parameters of the pathname supply related information. A logical name provides a name by which a resource is known to the job. The user can reference the logical name instead of the pathname or device name. A logical name also provides a mechanism for passing parameters associated with the resource.

A logical name may be global or job local in scope. Any user can assign a global logical name that is available to all system users. In contrast, a job-local logical name is available only to the specific job for which it is defined.

Logical names permit three extensions to the standard file types: job temporary files, logically concatenated files, and multivolume files. A logically concatenated file is a group of files known collectively by a single logical name. A multivolume file can exist on one or more disk volumes.

1.5.3.2 Job Temporary Files. Job temporary files are used only within the scope of a job. These temporary files are local to the job that is active when the files are created; they are deleted along with the job. Any task within the job can access a job temporary file.

To create a job temporary file, specify the parameters for the file on a Create Logical Name operation of the Name Management SVC; then, use the Assign LUNO operation of the I/O Operations SVC to assign a LUNO to the logical name, automatically creating the file.

Access to the file is by logical name. You can use a job temporary file to accumulate output data from multiple tasks or to pass data from one task to another.

The mechanism used to keep a job temporary file from being deleted gives the file the appearance of always having a LUNO assigned. For this reason, operations that require no LUNOs to be assigned to the file (that is, DF, MFN, or XE with exclusive use) will not succeed. A volume that has a job temporary file currently in use cannot be unloaded until the file has been deleted.

Job temporary files are implicitly deleted when the job terminates. They can also be deleted by releasing the logical name used to access the file. In the event of a system crash, temporary files are deleted at the next IPL.

1.5.3.3 File Concatenation. You can logically concatenate sequential and relative record files 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, you can physically concatenate the files via the Copy/Concatenate (CC) SCI command. A multifile set is a set of KIFs whose pathnames 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 can be on separate disks.

Several restrictions apply to the concatenation of files:

- The files must be of the same type.
- The files cannot be special use files such as directories, program files, KIFs, or image files.
- Relative record files to be concatenated must have the same logical record size.
- A concatenation of files cannot contain blocked and unblocked records.
- You must release any LUNO assigned to a file before concatenating the file.
- You cannot concatenate a file with itself.
- You cannot use a logical name at one site to concatenate files at another site.

Special rules apply to combining KIFs in a multifile set. At the first definition of the set, the following rules apply:

- All but the first file must be empty.
- No file can be a member of an existing multifile set.
- All files must have the same physical record size and the same key definitions.

In subsequent definitions of these sets, the following rules apply:

- The same files as in the first definition must be associated in the same order.
- You cannot omit any of the files that were in the original set.
- You can add one empty file at the end but not at any other position.

You can access a KIF of a multifile set only as an unblocked file.

A multifile set of KIFs permits a larger KIF than one disk can store. When a KIF can no longer expand because of insufficient space on the disk, you can create a new file on another disk. By using a logical name, you can use the two files as one. The second file is, in effect, an extension of the first. If the first file contains 5000 physical records and physical record 5001 is required, the first physical record of the second file, record 0, is used.

The following lists the file utility operations of the I/O Operations SVC that apply to concatenated and multivolume sets:

- Assign LUNO
- Release LUNO
- 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 you assign a LUNO to the logical name. You can access a concatenated file only for the duration of the logical name. You can specify files 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 you use the logical name.

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 (EOF) mark appears at the end-of-medium (EOM), the EOF is masked. This allows you to access concatenated files logically as a single file without receiving intermediate EOF marks. Note that any intermediate EOF mark not at the EOM is always returned. If you encounter two EOF marks at the EOM, a single EOF 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. An error occurs if any of the precreated files of a concatenated file are being accessed independently. This maintains file integrity. To delete a concatenated file, you must delete the individual files.

To back up individual components of a concatenated file or a multifile set, use the standard directory utilities. You cannot back them up by using the logical name. After backing up the files to a new medium, you can assign a new logical name to them and use them as before.

1.5.3.4 Multivolume File Capability. A multivolume file is a concatenated file or a multifile KIF set that is made up of files on two or more volumes. The same rules and limitations apply to concatenating files on different volumes as files on the same volume.

1.5.3.5 Stages of Name Definitions. All logical names associated with a given job are located in a unique segment in memory. Stages are used to divide the logical names within a name segment into independent groups. The same names can appear in more than one stage, but changing the value of a name in one stage does not affect the value of the name in any other stage.

Each stage within a name segment has a unique stage number between 0 and 255, and each task within a job is associated with a stage number. A task is not restricted to this stage. It can enter a new stage, and after some time executing there, it can return to its previous stage. Both of these operations are subopcodes of the Name Manager SVC. These operations are used by system software and are not documented with the standard user SVC.

When the first user logs on with a given user ID and job name, a name segment is created and initialized with the names retrieved from the file `.$USER.<USER ID>.SYN`. Each defined user ID has a directory under `.$USER`. The files in these directories are used to store name definitions while the user IDs are not being used and to recreate stage zero when a user logs on. Initially, only stage 0 exists. However, as soon as any foreground SCI task begins executing, it issues an Enter New Stage operation. The first SCI task that is bid in a new job initializes stage 1 with the names of stage 0 and associates SCI with the new stage.

When a user reconnects to an existing job, a similar set of events occurs. LOGON bids the new SCI under stage 0; SCI executes an Enter New Stage operation; a new stage number is allocated and initialized with the names of its parent stage (stage 0); and the new stage number is associated with the new SCI. Note that even though a user's disk-resident name file might change between the time that he logs on and the time that another user reconnects to his job, both users start execution with the same name definitions. The reason for this is that stage 0 is initialized only once and is never changed. However, if another user logs on with the same user ID and job name but does not reconnect, he begins execution with the current names in the synonym file since he is creating a new job whose name segment has not been initialized.

After SCI has finished initializing itself, a user can begin executing commands and programs. Any task which is bid by SCI and runs in the foreground begins executing with the same stage number as its parent task, SCI. This is true for all command processors that are bid by the .BID and .RBID primitives and for all user tasks that are bid via the Execute Task (XT) command. All command processors that are bid by the .QBID and .DBID primitives and all batch streams that are bid via the Execute Batch (XB) command begin execution in a new stage. SCI handles the latter case by executing an Enter New Stage operation and performing a regular bid. After the new task has begun executing in the new stage, SCI executes a Return to Previous Stage operation. A task bid in this fashion cannot change the name definitions of its parent stage.

When a user executes the Quit SCI (Q) command, SCI executes a Save Names operation. This operation causes the names associated with the stage of the requesting task to be saved in a specified file. For SCI, this file is always .\$\$USER.< USER ID> .SYN. Even if more than one user is reconnected to a job, this operation is performed after each user logs off. The file will reflect the name definitions of the last user to log off. Therefore, it is necessary for users to cooperate when sharing a a single name segment.

1.5.4 Interprocess Communication (IPC)

IPC provides communication between two or more tasks. Information passes through communication channels, and IPC is responsible for managing channel activity.

1.5.4.1 Channel Definition. A channel is a path through which data flows between two or more tasks. A single owner task controls each channel. One or more other tasks, called requesting tasks, can exchange data with the owner task. The scope of a channel can be global, job local, or task local, as follows:

Global Scope

A channel having global scope is potentially accessible by any task in the system. The owner task is nonreplicable and cannot be bid automatically by an Assign LUNO (AL) SCI command or a corresponding SVC. Multiple tasks can concurrently use a global channel that permits shared access.

Job-Local Scope

A channel having job-local scope is accessible by any task in the job. The owner task is replicable, one copy per job. The channel can be created as sharable, and the owner task can be bid automatically when the first AL command or corresponding SVC assigns a LUNO to the channel.

Task-Local Scope

A channel having task-local scope is accessible within a task. The owner task is replicated for each requester task. By definition, the channel is not sharable, and the owner task is bid automatically when the first AL command or corresponding SVC assigns a LUNO to the channel. Each requester task can independently assign a LUNO, open the channel, perform I/O, close the channel, and release the LUNO.

1.5.4.2 Channel Creation.

Creating a channel consists of the following steps:

1. Install the channel owner task in an existing program file or in a newly created program file. For a job-local or task-local channel, you must install the owner task as a replicatable task. The owner task of a global channel is not replicatable.
2. Create the channel by executing a Create IPC Channel (CIC) command or the corresponding SVC operation. This creates the channel with the specified characteristics.

1.5.4.3 Channel Characteristics. DNOS channels can be defined as symmetric channels or master/slave channels. Symmetric channels function with simple read and write requests, where one correspondent on the channel issues a read and another issues a write. The data written passes to the reader's buffer when a pair of requests match. In a master/slave channel, the master task receives the entire buffered SVC block for processing. The master (owner) task processes the block and returns the resulting block to the requesting (slave) task.

Symmetric Channel Activity. Symmetric channels communicate messages or data in a relatively restricted fashion. Tasks can be written to exchange information or facilitate use of common resources. The operations addressed to the channel by such tasks are limited to Open, Close, Read, Write, Read Status, and Write EOF.

When a task opens the channel, the access privileges requested are checked against those previously granted to other users of the channel. The same rules apply for channels as for other resources when granting or denying access. For example, if one requesting task opens a channel with privileges of exclusive all, no other task can open the channel. Shared access, which allows both Read and Write operations by all channels, is appropriate for most Open operations.

The channel definition can require more restrictive access privileges than a task specifies. A requesting task can open a symmetric channel established as a nonshared channel in any mode; however, the channel functions with only one task at a time. This is necessary for a symmetric channel since the owner task has no way of differentiating requesting tasks from each other.

Any of the following can issue a Close: an owner, a requesting task, or the system when processing an abnormal termination by a task on the channel. When a Close is issued by (or for) a requesting task, IPC processes the Close and the channel closes for that task.

On other operations to a symmetric channel, IPC must find a match before the operation is performed. That is, one task must issue a Read and the other a Write before either operation will be processed. If a mismatch occurs, both the requester and the owner tasks are informed of the error.

When an owner issues a Close, IPC processes the request and the channel is marked as dormant for all tasks for which it is currently open. This setting causes requesting tasks to receive errors on any operation except Open and Close. When a requesting task on a dormant channel issues a close, the channel becomes a closed channel for that task, and it is available to be opened.

Master/Slave Channel Activity. A master/slave channel is established when the owner task processes all requests from requesting tasks. The master/slave owner task can be written in assembly language using the Master Read, Master Write, and Read Call Block operations described in subsequent paragraphs; it can also be written in a high-level language with subroutine support to access these operations.

When accessing a master/slave channel, a requesting task may need to pass a set of parameters to the master task. The user can specify these parameters as part of an Assign Logical Name suboperation of the Name Manager SVC. Then, DNOS can pass them to the owner task as part of an AL command or a corresponding SVC.

To receive such parameters, an owner task may process Assign LUNO operations from requesting tasks. The owner receives the request with Master Read. The owner task then requests a Master Write of the assign block. Owner tasks that process assigns also process Release LUNO calls.

The owner task can process I/O Abort requests and all I/O Utility requests. You must specify these options in the Create IPC Channel (CIC) command.

While using a master/slave channel, the owner task processes I/O operations to a channel from requesting tasks. The owner task issues a Master Read to obtain a request for processing and issues a matching Master Write to return messages or status information to the requesting task.

IPC performs several operations for the master/slave channel as it does for the symmetric channel. In particular, IPC deals with an Open or Close issued by an owner task as described for symmetric channels. The owner Open specifies the channel access privileges in a manner consistent with I/O resources. The owner task processes Open and Close operations from a requesting task processed by the owner task. IPC modifies some internal counts to keep track of requesting task Open and Close operations when the owner executes a Master Write of the Open or Close block.

IPC passes to the owner task all I/O operations issued to the channel by the requesting task. The operations are processed accordingly. The owner task uses the following operations exclusively:

- Open
- Close
- Master Read
- Read Call Block
- Master Write
- Read Status

1.5.4.4 IPC Supervisor Calls (SVCs). The IPC operations are a subset of the operations that the I/O Operations SVC provides. In addition to those listed in the preceding paragraph for master tasks, the following operations apply to IPC channels:

- Create IPC Channel
- Delete IPC Channel
- Open
- Symmetric Read
- Symmetric Write
- Write EOF
- Close

1.5.4.5 IPC SCI Commands. The following SCI commands support IPC capabilities:

Create IPC Channel (CIC)

The CIC command creates a global, job-local, or task-local channel. A global channel is accessible by any task in the system, and the owner task is nonreplicable. A job-local channel is accessible by any task in the job, and the owner task is replicatable. The owner task of a task-local channel is replicatable for each task in any job.

Delete IPC Channel (DIC)

The DIC command deletes the disk-based definition of the channel specified.

Show Channel Status (SCS)

The SCS command displays information about a specified active channel. This information includes channel owner, type of channel, scope of channel, maximum message length, shared or not shared scope, number of current assigns, number of current opens, and current access privileges.



Disk and File Organization

2.1 FILE ORGANIZATION

DNOS provides disk file support for applications and system programs. A file is a named and organized collection of records. Disk files are written on any of several disk media used with DNOS. You can access the files through the I/O subsystem. The following paragraphs describe the types of files, ways of protecting and sharing them, and their characteristics.

2.1.1 File Types

DNOS supports three types of disk files:

- Sequential files
- Relative record files
- Key indexed files (KIFs)

Relative record files include three special usage groups:

- Directory files
- Program files
- Image files

2.1.1.1 Sequential Files. In a sequential file, the order in which the records are written determines record organization. You cannot alter the record sequence by adding or deleting records except in the following cases:

- You can add records in sequence following the last record in the file.
- You can rewrite a record if the record length does not change.

On a blank-suppressed file, the blank-suppressed record length must not change during a rewrite operation; the internal size of the record must be the same. Records in a sequential file are of variable length, and you access the records serially (record 0 first, record 1 next, and so on). Records are accessed in the order in which they were originally written.

Encountering an end-of-file (EOF) on a read of a sequential file indicates that the file is positioned after the last record.

2.1.1.2 Relative Record Files. A relative record file consists of records that are identified by position. In effect, the file is a string of logical records, each accessed by a record number. The first logical record is record 0. Therefore, to access the tenth record, you should enter 9 in the appropriate field of the I/O supervisor call (SVC) block. You can access relative record files sequentially by placing a starting value in the record number field of the I/O SVC block. DNOS automatically increments the record number after each read or write. The range of record numbers is from 0 to one less than the number of records in the file. The upper limit is 16,777,216. Records in a relative record file are of fixed length. The length is specified when the file is created.

DNOS converts the record number to a physical address on the disk (track and sector). It can directly access any record with one disk access.

Relative record files can be blocked or unblocked. Generally, blocking allows faster processing. You can delay actual disk transfers of memory buffers for blocked relative record files. Once a buffer for a block is allocated in memory and the block is read from disk, all Read operations from that block reference the memory buffer for the block. Unless you select the immediate-write option, information directed to records already in memory is not written back to disk until DNOS requires the memory space allocated to the blocking buffer or until the file is closed.

When DNOS reads an EOF on a relative record file, the record number is used but not incremented in the SVC block.

2.1.1.3 Key Indexed Files (KIFs). A KIF consists of data records that you can access by content. You can define various fields within a record as a key. Each record can have up to 14 keys, with access through each key independent of the other keys. For example, the records in an employee file might be accessed by employee ID, employee name, and employee social security number.

In addition to random access by key value, KIFs have the following features:

- You can access records sequentially in the sort order of any key.
- At file creation, you can give a key the attribute of allowing duplicates (that is, of allowing two or more records in the file with the same value for this key).
- At file creation, you can give a key the attribute of being modifiable. This allows you to change the key value when you write a record. Also, a modifiable key value can be missing in the record but added later on a rewrite. Note that you cannot assign this attribute to the first (primary) key.
- Key fields can overlap if their attributes match.
- A key can be up to 100 contiguous characters long.
- Records can be of variable length and can change in size on a rewrite.
- Positioning on partial keys is allowed.
- Records are automatically blank compressed.
- Record-level locking (temporary exclusive-all access) is supported.

- The file can grow in size.
- Preimage logging of modified blocks maintains file integrity. As a result, system crashes and power failures cause the loss of only the last I/O operation.
- A KIF cannot contain records of odd or zero length.
- The EOF on a KIF is analogous to the EOF on a relative record file.

2.1.1.4 Special Usage Files. DNOS supports three special types of relative record files:

Directory File

Contains information necessary to locate other files and descriptive information about those files. It does not contain user data.

Program File

Contains executable programs or segments in memory image form. A program file usually contains more than one program.

Image File

Has a logical record size that equals the physical record size, which in turn equals the disk sector size. Image files usually contain a memory image of some code. These files are designed so that a program image can be read into memory in one disk access.

2.1.2 File Protection and Sharing

DNOS ensures disk file integrity and allows you to control the use and modification of files by means of the following features:

- Delete and write protection
- Record locking
- Access privileges
- Special usage file protection

2.1.2.1 Delete and Write Protection. Standard I/O calls modify the delete and write protection file attributes. Files are initially created without protection. You must make a subsequent I/O call to change the protection status.

An attempt to write to or delete a write-protected file fails and returns an error code. (Write protection includes delete protection.) These protective attributes are not intended for file security. (Nonprivileged SVCs are available to remove write and delete protection.) Instead, they provide protection against user error or program flaws that might otherwise destroy valuable data.

2.1.2.2 Record Locking. Record locking restricts access to a record in a file. This means that although several users share access to a given file, you can lock individual records within the file to provide exclusive (single-user) read and write access. This is not a security feature, since any file user can unlock a locked record; however, this feature can ensure that record updates occur one at a time. For example, inventory files can be accessible from several terminals. Record locking can prevent two or more users from attempting to update a record simultaneously, causing an undetected loss of one of the updates.

2.1.2.3 Access Privileges. To assist intertask I/O synchronization, DNOS supports several different access modes for all I/O resources. These modes define the relationship between logical units and resources and prevents conflicting accesses by other logical units. Four access modes apply to files. Enforcement of file access privileges is through the Open, Open Rewind, and Open Random operations of the I/O Operations SVC. The SVC fails if you request an operation with a conflicting privilege level. You can change access privileges if no access conflicts result.

With respect to access privileges, a Write operation is any operation that alters the contents of a file. The access privileges, which conform to the American National Standards Institute (ANSI) standard, are as follows:

- **Read-Only** — Allows the calling program to read but not write. Gives other programs read-only, shared, and exclusive write access.
- **Shared** — Allows the calling program to read and write. Gives other programs read-only and shared access.
- **Exclusive write** — Allows the calling program to read and write. Allows other programs to read but not write.
- **Exclusive all** — Allows the calling program to read and write. Does not allow other programs to have access.

The Open, Open Rewind, Open Random, and Modify Privileges operations use bits 3 and 4 of the user flags in the I/O SVC block to select the following access privileges:

Code	Meaning
00	Exclusive write
01	Exclusive all
10	Shared
11	Read only

2.1.2.4 Special Usage File Protection. To prevent accidental use of special usage files (program, directory, and image files) as data files, you must set two flags in the I/O SVC block for the Assign LUNO operation. These flags indicate whether the LUNO is being assigned to a program file, a directory, an image file, or a file with no special usage. You must set the proper flags to set the LUNO; otherwise, an error code is returned. The flags are bits 1 and 2 in byte 16 of the I/O Operations SVC block. For further details, refer to the *DNOS Supervisor Call (SVC) Reference Manual*.

2.1.3 File Characteristics

The following paragraphs describe these characteristics of DNOS files:

- Record blocking
- Saving disk space
- Immediate write
- Temporary attribute
- Expandability
- End-of-file

2.1.3.1 Record Blocking. A file consists of a collection of data entities called logical records. The logical records do not necessarily correspond to the physical records (that is, to the physical division of data on the disk). Logical records are the data groupings of a file as seen by a program. Physical records are the buffers physically transferred between memory and disk.

The length of the logical records within a file can be constant or variable, depending on the file type. For relative record files, logical records are of fixed length. This makes it possible for the system to calculate the physical position of any logical record relative to the beginning of the file. This characteristic makes possible random access of a logical record in relative record files.

Sequential files and KIFs allow variable-length logical records. For KIFs, the logical record length is always an even number of bytes. For sequential files, the logical records can be any number of bytes, including zero.

When you create a file, you must specify the logical record size. For relative record files, the size must be exact. For sequential files and KIFs, the record size is used to calculate the amount of disk space initially allocated to the file; the specified size should be an estimate of the average logical record size. The more accurate the estimate, the better the utilization of disk space.

The physical record length is specified when the file is created and cannot be changed.

It is often advantageous to store multiple logical records in a physical record. This is called blocking.

Since disk transfer and latency times are relatively long, usually you should choose physical records large enough to include several logical records. When a task first issues a read request, DNOS actually reads an entire physical record 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 requests to a physical record in memory do not cause immediate disk access; instead, they reference the record image in memory. DNOS keeps an accessed physical record, which usually contains several logical records, in memory until the memory area is needed for other purposes or the file is closed. Blocking logical records and the deferred write capabilities can substantially improve system throughput, especially for sequential files.

If no physical record size is specified at creation time, DNOS assigns a default physical record size based on the file type. Sequential files and KIFs support variable-length records. Since DNOS can split logical records into two or more physical records in a sequential file, the physical record size can be smaller than the largest logical record. However, this results in inefficient processing of the file.

2.1.3.2 Saving Disk Space. Blank suppression and blank adjustment are two methods of saving disk space within a file by storing data in a more compact form. These methods apply only to KIFs and sequential files. While blank suppression always occurs on KIFs, it is optional on sequential files.

Blank suppression 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 suppression is transparent to the user. Usually, you should specify blank suppression for a source file, a listing file, or a text file, since these files tend to contain many blanks. However, keep in mind that blank suppression increases by one word the length of records containing no blanks.

The second method, blank adjustment, applies to sequential files and I/O devices with variable record lengths. Blank adjustment truncates trailing blanks on output and restores them on input. To use this feature, you must set the blank adjust flag bit in an I/O SVC block.

2.1.3.3 Immediate Write. When DNOS writes a record of a blocked file, the record is placed in a blocking buffer in memory. The record in the buffer remains in memory as long as possible. Subsequent read and write requests access the memory buffer and not the disk. The disk is accessed only when DNOS needs the memory occupied by the blocking buffer or the file is closed. This delaying of disk writes increases system throughput. However, although the disk write is actually delayed, it is reported as being complete. Consequently, errors that occur during the write cycle are unexpected and in some situations may not be detected. DNOS supports an immediate-write option, specified when a file is created, for files that cannot risk an undetected write error.

The most common undetected error is disk failure. For example, you might update a record in a block and be informed that the update has been successfully completed. However, when the block is actually written to disk, possibly several minutes later, an I/O error might occur. This error is returned on the next call to the LUNO after the error. The error is returned even if the call is not a Write operation.

When deciding whether to include the immediate-write option, remember that undetected errors are rare and that files (especially sequential files) with this option are less efficiently processed. Therefore, you should reserve this option for sensitive files that cannot risk loss of data. KIFs always include the immediate-write option.

2.1.3.4 Temporary Attribute. When you create a file, it usually remains in existence until you explicitly delete it. However, under DNOS you can also create temporary files as follows:

- Create a temporary file using an Assign LUNO operation of the I/O SVC with the temporary file bit set. In this case, the file remains in existence only as long as the LUNO is assigned. If you do not specify a name, DNOS assigns a unique temporary file name. However, the pathname portion of the I/O SVC block can indicate the disk volume on which the file is to be created. By using the Rename File operation of the I/O Operations SVC, you can explicitly name the file and specify it as permanent. Otherwise, it is deleted the first time its LUNO is released.
- Create a temporary file using a Create File operation of the I/O Operations SVC with the temporary bit set and a pathname supplied. You can assign one or more LUNOs to the file using the pathname. The file remains in existence as long as at least one LUNO is assigned. When the last LUNO is released, the file is deleted.
- Create a job-temporary file by using the Assign Logical Name SVC or the Assign Logical Name (ALN) SCI command followed by either the Create File (CF) command or the Assign Luno (AL) command. The discussion of name management in Section 1 describes how to create a job-temporary file using these commands.

2.1.3.5 Expandability. When you create a file using the Create File operation of the I/O Operations SVC, you can give the file a fixed size via the primary allocation parameter. Alternatively, you can create the file as expandable and give the primary allocation as its initial allocation. When the file exceeds its primary allocation, it is augmented with secondary allocations. The secondary allocation parameter is the size of the first secondary allocation. Subsequent secondary allocations automatically and progressively increase in size over the previous allocation. Files add up to a maximum of 16 secondary allocations.

2.1.3.6 End-of-File (EOF). An EOF is a logical position within a relative record file or KIF. It is an actual record within sequential files. When read, it sets the EOF status bit. 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 can have more than one EOF. A sequential file is analogous to a reel of magnetic tape that can contain several files separated by EOFs. A sequential file can consist of multiple data sets or subfiles marked by EOFs. A KIF has a logical EOF that corresponds to the record following the record with the largest primary key. For a KIF, the EOF applies only to Read ASCII operations and Forward Space operations that access the file sequentially in primary key order.

The internal representation of an EOF in a sequential file is a record of zero length. Either a Write EOF operation or a Close and Write EOF operation writes the EOF. Writing an EOF does not prevent writing more records to the file.

2.2 DISK ORGANIZATION

The organization of files on the disk is related to the following:

- Disk characteristics
- Allocation of space on the disk
- Physical organization of the disk

2.2.1 Disk Characteristics

All tracks on disks are initialized in a one-sector-per-record format. This record size is a characteristic of the type of disk and is not necessarily the physical record size for files to be created on the disk.

Disks are logically divided into allocatable disk units (ADUs). An ADU is made up of one or more complete sectors of the disk. The number of sectors per ADU varies according to the disk type (see Table 2-1) to provide a number of ADUs per disk less than 65,536. Each ADU on the disk can be addressed by a 16-bit word. ADU numbers start with 0, the first ADU starting on track 0, sector 0.

Table 2-1. Format Information for Supported Disks

Disk Type	Available Space (M Bytes)	No. ADUs	No. Heads	No. Cylinder	Sec./Track	Sec./ADU	Bytes/Sec.
DS10	4.7	16,320	2	408	20	1	288
DS25	22.3	25,840	5	408	38	3	288
DS50	44.6	51,616	5	815	38	3	288
DS80	62.7	40,819	5	803	61	6	256
DS200	169.5	65,381	19	815	38	9	288
DS300	238.3	62,045	19	803	61	15	256
FD1000	1.15	4,004	2	77	26	1	288
CD1400/32 Removable	13.5	52,544	1	821	64	1	256
Fixed	13.5	52,544	1	821	64	1	256
CD1400/96 Removable	13.5	52,544	1	821	64	1	256
Fixed	67.3	43,786	5	821	64	6	256
WD500	4.9	19,200	4	150	32	1	256
WD500A	17.0	22,208	3	694	32	3	256
WD800-18	18.5	24,087	3	651	37	3	256
WD800-43	43.2	56,203	7	651	37	3	256
WD800A/38	38.4	50,105	5	911	33	3	256
WD800A/68	69.2	45,094	9	911	33	6	256
WD800A/114	114.5	49,720	15	904	33	9	256
WD900-138	138.1	59,928	10	805	67	9	256
WD900-138/2	69.0	44,946	5	805	67	6	256
WD900-425	425.8	61,600	24	693	100	27	256
WD900-425/2	212.9	55,440	12	693	100	15	256

2.2.2 Disk Space Allocation to Files

Disk space is allocated to files in multiples of ADUs. The ADU size, physical record length, and logical record length determine how efficiently disk space is utilized. Consider the following disk access characteristics:

- Physical records start on sector boundaries.
- Physical records that do not start on an ADU boundary cannot span an ADU boundary.
- Logical records can span physical record boundaries in sequential files only.

For efficient utilization of disk space by a file, the physical record size should be an integer multiple of the sector size and an integer multiple or a factor of an ADU. If the file is a relative record file, the physical record size should be an integer multiple of the logical record size.

An additional consideration in file definition is frequency of disk access. A disk access is required only when an I/O operation addresses a record that is not in the buffered physical record. As a rule, the physical record length should be at least three times the logical record length, allowing file management to buffer logical records.

2.2.3 Physical Organization of a DNOS Disk

Disks initialized under DNOS have the following physical layout:

- Track 0/Sector 0 — Contains volume information such as the volume name and the location of VCATALOG.
- Track 0/Sector 1 — Contains bad ADU list.
- Remainder of track 0 — Contains bit maps indicating disk allocation information. The largest available block is recorded at the beginning.
- Track 1 — Contains the disk program image loader and a copy of sectors 0 and 1 of track 0, used for recovery from a major disk failure.
- Remaining tracks — Available for file allocation.
- Reserved tracks — Contain alternate location of bad tracks on disks that support bad track mapping.

2.2.3.1 Volume Information. The information stored in track 0, sector 0 of all disks initialized under DNOS is called *volume information*. Figure 2-1 shows the format of this 164-byte block of information. In this figure and those that follow, reserved fields are fields that DNOS does not currently use but might use in the future.

DEC	HEX	VOLUME NAME	
0-7	0-7	NUMBER OF ADUS	
8-9	8-9	BIT MAP SECTOR No.	No. OF BIT MAPS
10-11	A-B	BYTES PER PHYSICAL RECORD	
12-13	C-D	PROGRAM IMAGE LOADER TRACK NUMBER	
14-15	E-F	RESERVED	
16-21	10-15	NUMBER OF BAD ADUS	
22-23	16-17	PROGRAM IMAGE LOADER ENTRY POINT	
24-25	18-19	LENGTH OF PROGRAM IMAGE LOADER	
26-27	1A-1B	RESERVED	
28-35	1C-23	PROGRAM IMAGE LOADER TRACK NUMBER	
36-37	24-25	RESERVED	
38-45	26-2D	PRIMARY SYSTEM IMAGE FILE NAME	
46-53	2E-35	SECONDARY SYSTEM IMAGE FILE NAME	
54-61	36-3D	SYSTEM IMAGE SELECT FLAG	
62-63	3E-3F	VCATALOG STARTING ADU	
64-65	40-41	VCATALOG PHYSICAL RECORD SIZE	
66-67	42-43	SECTORS/ADU	
68-69	44-45	CREATION DATE	
70-73	46-49	PRIMARY PROGRAM FILE NAME	
74-81	4A-51		

2279387 (1/2.)

Figure 2-1. Volume Information Format (Sheet 1 of 2)

DEC	HEX	
82-89	52-59	SECONDARY PROGRAM FILE NAME
90-91	5A-5B	PROGRAM FILE SELECT FLAG
92-99	5C-63	PRIMARY OVERLAY FILE NAME
100-107	64-6B	SECONDARY OVERLAY FILE NAME
108-109	6C-6D	OVERLAY FILE SELECT FLAG
110-117	6E-75	PRIMARY SYSTEM LOADER FILE NAME
118-125	76-7D	SECONDARY SYSTEM LOADER FILE NAME
126-127	7E-7F	SYSTEM LOADER SELECT FLAG
128-135	80-87	DIAGNOSTIC FILE NAME
136-137	88-89	DIAGNOSTIC SELECT FLAG
138-143	8A-8F	RESERVED
144-151	90-97	WRITABLE CONTROL STORAGE FILE NAME
152-159	98-9F	WCS SECONDARY FILE
160-161	A0-A1	SELECT SWITCH
162-163	A2-A3	TRACK 1 SELECT FLAG

2279387 (2/2)

Figure 2-1. Volume Information Format (Sheet 2 of 2)

The volume information shown in Figure 2-1 contains the following field descriptions:

Byte	Description
0 – 7	Volume name, one to eight characters, blank filled to the right.
8 – 9	Total number of ADUs contained in the volume. This field varies by disk type.
10	The number of the sector in track 0 in which the first bit map resides.
11	Total number of bit maps.
12 – 13	The number of bytes per physical record (that is, sector) in track 0. This value is also disk dependent.
14 – 15	The number of the track that contains the disk program image loader. This field is initialized to 1.
16 – 21	Reserved.
22 – 23	Total number of bad ADUs on the disk.
24 – 25	Entry point address of the disk program image loader (initialized to > A4, the entry point of the loader when it is loaded at location > A0).
26 – 27	Total byte length of the disk program image loader.
28 – 35	Reserved.
36 – 37	Second copy of the track that contains the disk program image loader (initialized to 1).
38 – 45	Reserved.
46 – 53	Name of the primary system image file (one to eight characters). Zero at initialization.
54 – 61	Name of the secondary system image file. Zero at initialization.
62 – 63	System select flag. Zero at initialization.
64 – 65	Number of the ADU in which the volume directory (VCATALOG) begins.
66 – 67	Physical record size of the VCATALOG directory file.
68 – 69	Number of sectors per ADU (disk dependent).
70 – 73	Disk creation date.

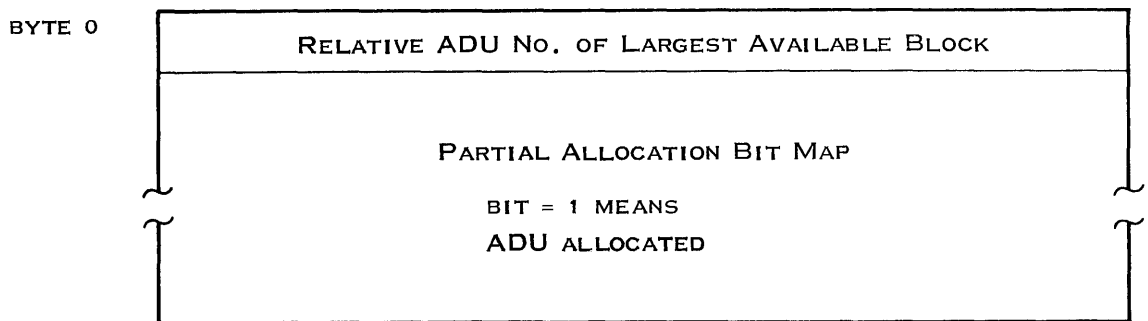
NOTE

The remaining fields of the volume information block apply to system disks only. They are not given values when the disk is initialized. The Modify Volume Information (MVI) command writes the field values.

Byte	Description
74 – 81	Primary system program file name
82 – 89	Secondary system program file name
90 – 91	System program file select flag
92 – 99	Primary system overlay file name
100 – 107	Secondary system overlay file name
108 – 109	System overlay file select flag
110 – 117	Primary system loader file name
118 – 125	Secondary system loader file name
126 – 127	System loader select flag
128 – 135	Diagnostic file name
136 – 137	Diagnostic select flag
138 – 143	Reserved
144 – 151	Writable control store (WCS) file name
152 – 159	WCS secondary file
160 – 161	Select switch
162 – 163	Track 1 select flag

2.2.3.2 Bit Map. To identify which areas on the disk are allocated and which are free, DNOS maintains a bit map of allocated ADUs. The bit map is located in track 0 of each disk, starting at sector 2 and continuing through as many sectors as necessary.

The bit map is divided into 128-word partial bit maps. Each partial bit map is located in a separate sector in track 0. The first word of each partial bit map contains the number of the ADU that begins the largest block of free disk space located in that part of the disk, which is mapped by the partial bit map. Each bit in the remaining 127 words represents an ADU. If the bit is zero, the ADU is free; if it is one, the ADU is allocated (or the ADU is on a bad track). Each partial bit map contains 127 16-bit words of information and maps 2032 ADUs. Figure 2-2 shows the structure of a partial bit map.



2279388

Figure 2-2. Partial Bit Map

2.2.4 Displaying and Modifying Absolute Disk Addresses

The following SCI commands are available to display or modify absolute disk addresses:

Command	Description
SAD	Show Absolute Disk
SADU	Show Allocatable Disk Unit
MAD	Modify Absolute Disk
MADU	Modify Allocatable Disk Unit

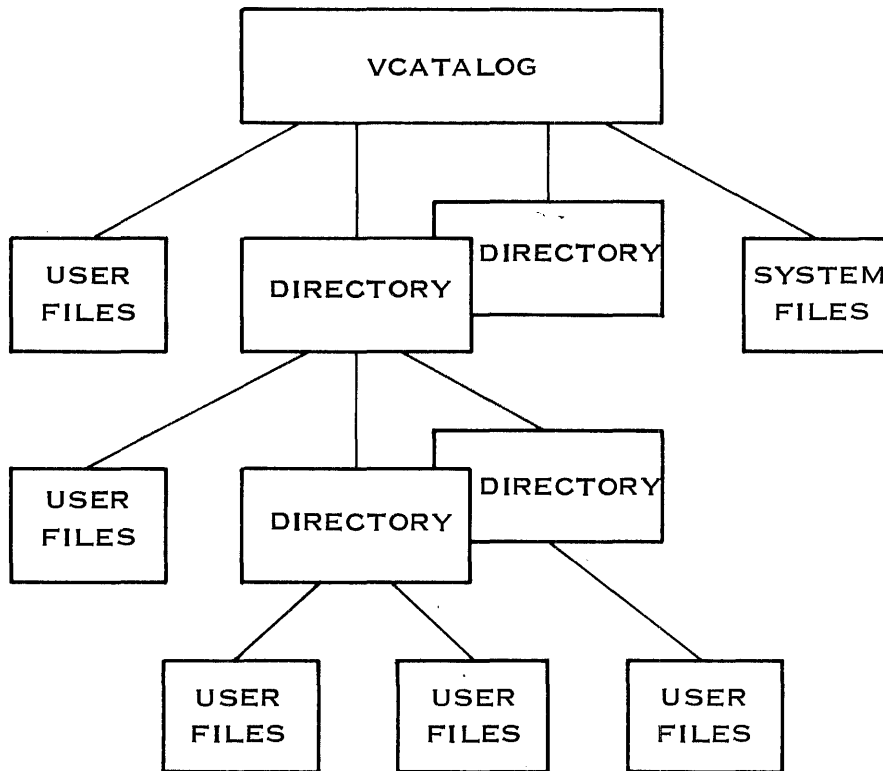
2.3 DISK FILE STRUCTURES

The structure of the directory file is a key to the organization of files on a disk. The following paragraphs describe the directory structure and the structure of each type of file that DNOS supports.

2.3.1 Directory File

A directory contains information necessary to locate other files and descriptions of those files.

Figure 2-3 illustrates the way in which all directories are connected in a network. The top of this network is the volume directory, called VCATALOG. VCATALOG is created on each volume when the disk is initialized. It maintains information about directories, system files, and user files.



2279389

Figure 2-3. Directory Structure

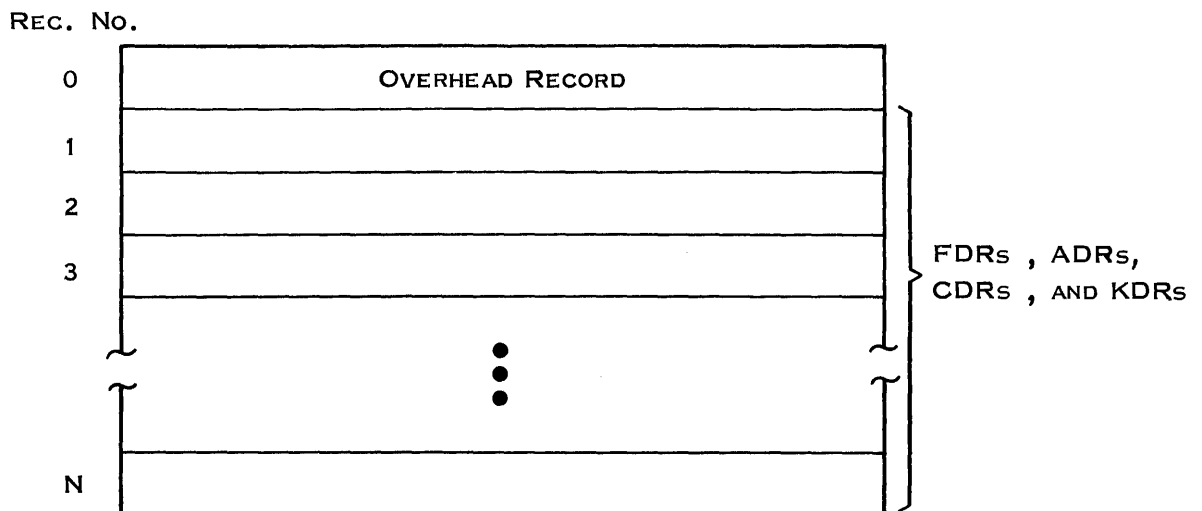
2.3.1.1 Directory File Characteristics. Directory files are unblocked relative record files consisting of one logical record per physical record. Figure 2-4 shows the file structure of a directory. Record 0 contains overhead information in the format shown in Figure 2-5. Each of the remaining records is of one of the following types:

- File descriptor record (FDR) — Describes a file and its location on the disk.
- Alias descriptor record (ADR) — Describes an alias for a file, includes the location of the file, and points to the FDR of the file.

- Channel descriptor record (CDR) — Describes a channel, specifies the program file of the owner task of the channel, and points to the FDR of the program file.
- Key descriptor record (KDR) — Describes the keys defined for a KIF. An entry in the FDR of the KIF points to the KDR. Thus, each KIF requires two directory entries.

Subsequent paragraphs describe the types of records in a directory.

File names in a directory are hashed to a record number, 1 through N, where N is the last record in the directory. If a file name hashes to a record number and the record is unused, an FDR for the file being inserted is built in that record. If the record is already used, a linear search from the hashed record finds a free record. For KIFs, a linear search is performed from the FDR to locate an available record for the key descriptor block.



2279390

Figure 2-4. Directory File Structure

DEC	HEX	
0-1	0-1	NUMBER OF RECORDS IN DIRECTORY
2-3	2-3	NUMBER OF FILES IN DIRECTORY
4-5	4-5	NUMBER OF AVAILABLE RECORDS
6-7	6-7	NO. OF TEMP. FILES CURRENTLY DEFINED
8-15	8-F	FILE NAME OF DIRECTORY
16-17	10-11	LEVEL NUMBER OF DIRECTORY
18-25	12-19	FILE NAME OF PARENT
26-63	1A-3F	RESERVED

2279391

Figure 2-5. Directory Overhead Record Format

The directory overhead record, record 0 of all directories, contains the following:

- Maximum number of records (entries) in the directory
- Number of currently defined files
- Number of available records (entries)
- File name of the directory
- Level number of the directory in the disk hierarchy (VCATALOG is level 0)
- File name of the parent directory

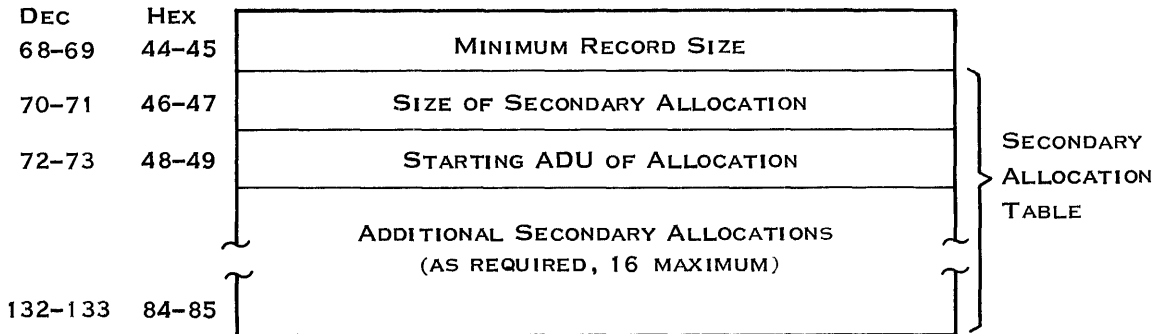
2.3.1.2 File Descriptor Record (FDR). Each file cataloged under the directory is represented by an FDR. Figure 2-6 shows an FDR.

DEC	HEX	
0-1	0-1	HASH KEY COUNT
2-3	2-3	HASH KEY
4-11	4-B	FILE NAME
12-15	C-F	RESERVED
16-17	10-11	FLAGS
18-19	12-13	PHYSICAL RECORD SIZE
20-21	14-15	LOGICAL RECORD SIZE
22-23	16-17	PRIMARY ALLOCATION SIZE
24-25	18-19	PRIMARY ALLOCATION ADU
26-27	1A-1B	SECONDARY ALLOCATION SIZE
28-29	1C-1D	OFFSET TO SECONDARY ALLOCATION TABLE
30-31	1E-1F	RECORD NUMBER OF FIRST ALIAS
32-35	20-23	END OF MEDIUM LOGICAL RECORD NUMBER
36-39	24-27	END OF MEDIUM BLOCK NUMBER
40-41	28-29	END OF MEDIUM OFFSET
42-45	2A-2D	FREE BLOCK QUEUE HEAD
46-47	2E-2F	BLOCK No. OF B-TREE ROOT (PRIMARY KEY)
48-49	30-31	BLOCK NUMBER OF FIRST DATA BLOCKS
50-51	32-33	TOTAL NUMBER OF DATA BLOCKS
52-53	34-35	RECORD NUMBER OF KEY DESCRIPTORS
54-59	36-3B	DATE AND TIME OF LAST UPDATE
60-65	3C-41	DATE AND TIME FILE CREATION
66	42	ADUs /BLOCK BLOCKS/ADU

KIF
FILES
ONLY

2279392 (1/2)

Figure 2-6. FDR Format (Sheet 1 of 2)



2279392 (2/2)

Figure 2-6. FDR Format (Sheet 2 of 2)

The FDR shown in Figure 2-6 contains the following information:

Byte	Description
0 – 1	Hash key count. The number of records in the directory that hashed to this record number.
2 – 3	Hash key. The result of the hash algorithm for the file name in this FDR. The hash value might not be the number of this record. When the hash value record has already been written, DNOS searches linearly for an unused record.
4 – 11	File name (eight characters).
12 – 15	Reserved.
16 – 17	File usage flags.
18 – 19	Physical record size in bytes. Must be an even number.
20 – 21	Logical record size in bytes. Must be an even number if the file is unblocked.
22 – 23	Primary allocation size, in ADUs.
24 – 25	Primary allocation starting ADU number (starting disk address).
26 – 27	Secondary allocation size in ADUs.
28 – 29	Offset into this FDR of the secondary allocation table. When the file is not expandable or before a secondary allocation has been made, the field contains 0.

Byte	Description
30 – 31	Record number of the ADR for the file's first alias or of the CDR. Contains zero when no alias is defined and no CDR exists.
32 – 35	Logical record number of the end-of-medium (EOM). The EOM is the end of the last space allocated to the file.
36 – 39	The logical block (physical record) number of the EOM.
40 – 41	The offset into the EOM block of the logical record following the EOM record.
42 – 45	Block number of the first block in a queue of KIF blocks with available space. Each block points to the next block in the queue. A block is a physical record of the file. This number is used only for KIFs.
46 – 47	The block number of the B-tree root block of the primary key. The block following this is the KIF root block for key 2, and so on. This field is also the total number of blocks that can be used for logging.
48 – 49	The block number of the first KIF data block.
50 – 51	The total number of data blocks in the KIF.
52 – 53	Record number of the KDR.
54 – 59	Date of the last update to the file.
60 – 65	Creation date of the file.
66	The number of ADUs per physical record.
67	The number of physical records per ADU.
68 – 69	The minimum size for a KIF logical record; the KIF must contain all of the keys defined.
70 – 133	The secondary allocation table, which contains two-word entries. The first word of an entry contains the size, in ADUs, of the first secondary allocation. The second word contains the starting ADU of the allocation. The table can contain as many as 16 entries and is used only when the file expands. Unused fields contain zeros.

The file usage flags in bytes 16 and 17 have the following meanings:

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

2279394

Bit(s)	Meaning
0-1	File usage, as follows: 00 — No special usage 01 — Directory file 10 — Program file 11 — Image file
2-3	Format, as follows: 00 — Binary 01 — Blank compressed
4	Allocation type, as follows: 1 — Expandable 0 — Primary allocation only
5-6	File type, as follows: 01 — Sequential 10 — Relative record 11 — Key indexed
7	Write protected, as follows: 1 — Write protected 0 — Not write protected
8	Delete protected, as follows: 1 — Delete protected 0 — Not delete protected
9	Temporary file, as follows: 1 — Temporary file 0 — Not a temporary file
10	Blocked, as follows: 1 — Unblocked 0 — Blocked

Bit(s)	Meaning
11	Reserved
12	Immediate write, as follows: 1 — Immediate write mode 0 — Deferred write mode
13–14	Reserved
15	Reserved

2.3.1.3 Alias Descriptor Record (ADR). An alias is an alternate name for a file. A directory contains an ADR for each alias of any file in the directory. The assignment of a record number for an ADR is similar to the assignment of a record number for an FDR. The alias is hashed to derive a record number. When the record is available, the ADR is written to that record. Otherwise, DNOS searches the file linearly from that record to locate an available record; DNOS writes the ADR in the first available record.

The number of aliases a file can have is limited only by the number of empty records available in the directory. An ADR implements each alias. The ADRs for the aliases of a file are linked to the FDR of the file and to each other.

The program file of a task that is the owner task of an IPC channel has one or more CDRs linked to the FDR of the file along with any ADRs associated with the file.

Figure 2-7 shows the format of the ADR, which is similar to that of the FDR. It includes 34 bytes. A flag identifies the record as an ADR. A field of the ADR contains the record number of the FDR for the file. Another field contains the record number of the next ADR or CDR linked to the FDR. When no record is linked to the ADR (that is, this ADR is the end of the linked list), this field contains 0.

DEC	HEX	
0–1	0–1	HASH KEY COUNT
2–3	2–3	HASH KEY
4–11	4–B	ALIAS
12–15	C–F	RESERVED
16–17	10–11	FLAGS
18–29	12–1D	RESERVED
30–31	1E–1F	RECORD NUMBER OF NEXT ADR OR CDR
32–33	20–21	RECORD NUMBER OF FDR

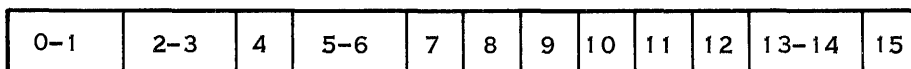
2279393

Figure 2-7. ADR Format

The ADR shown in Figure 2-7 contains the following information:

Byte	Description
0 – 1	Hash key count. The number of records in the directory that hashed to this record number.
2 – 3	Hash key. The result of the hash algorithm for the alias in this ADR. The hash value might not be the number of this record. When the hash value record has already been written, DNOS searches linearly for an unused record.
4 – 11	Alias. The alias in this item is an alternate name for the file (that is, a secondary name by which a previously defined file is also known). The primary name for a file is supplied in the FDR. Secondary names are documented in the ADR.
12 – 15	Reserved.
16 – 17	File usage flags. These apply to the file and are identical to those in the FDR except that bit 11 is set to identify this record as an ADR.
18 – 29	Reserved.
30 – 31	Record number of next alias. This is a pointer chaining forward to another ADR for the same file, if one exists. If one does not exist, this value is 0.
32 – 33	Record number of actual file. A pointer to the directory file record that contains the file descriptor for this particular file.

The file usage flags in bytes 16 and 17 apply to the file described in the FDR at the record in bytes 30 and 31. The flags have the following meanings:



2279395

Bit(s)	Meaning
0 – 1	File usage, as follows: 00 — No special usage 01 — Directory file 10 — Program file 11 — Image file
2 – 3	Format, as follows: 00 — Binary 01 — Blank compressed

Bit(s)	Meaning
4	Allocation type, as follows: 1 — Expandable 0 — Primary allocation only
5 – 6	File type, as follows: 01 — Sequential 10 — Relative record 11 — Key indexed
7	Write protected, as follows: 1 — Write protected 0 — Not write protected
8	Delete protected, as follows: 1 — Delete protected 0 — Not delete protected
9	Temporary file, as follows: 1 — Temporary file 0 — Not a temporary file
10	Blocked, as follows: 1 — Unblocked 0 — Blocked
11	ADR; set to 1.
12	Immediate write, as follows: 1 — Immediate write mode 0 — Deferred write mode
13 – 14	Reserved
15	Reserved; set to 0.

2.3.1.4 Channel Descriptor Record (CDR). The CDR describes an IPC channel. It is associated with the program file of the channel owner task and is linked to the FDR of the program file along with any aliases for the file.

The allocation of a record in the directory for a CDR is similar to the allocation of an ADR. The channel name is hashed and the result is used as a record number. When the record is already occupied, the next available record is used.

Figure 2-8 shows the format of the CDR.

DEC	HEX	HASH KEY COUNT	
0-1	0-1	HASH KEY	
2-3	2-3	CHANNEL NAME	
4-11	4-B	RESERVED	
12-15	C-F	FLAGS	
16-17	10-11	CHANNEL FLAGS	INSTALLED ID
18	12	DEFAULT RESOURCE	RESOURCE TYPE FLAGS
20	14	MAXIMUM MESSAGE LENGTH	
22-23	16-17	RESERVED	
24-29	18-1D	RECORD NUMBER OF NEXT CDR OR ADR	
30-31	1E-1F	RECORD NUMBER OF FDR	
32-33	20-21	RESERVED	
34-143	22-8F	USER ID	RESERVED
144	90	RESERVED	
146-255	92-FF	RESERVED	

2279396

Figure 2-8. CDR Format

The CDR shown in Figure 2-8 contains the following information:

Byte	Description
0 - 1	Hash key count. The number of records in the directory that hashed to this record number.
2 - 3	Hash key. The result of the hash algorithm for the channel number. The hash value might not be the number of this record. When the hash value record has already been written, DNOS searches linearly for an unused record.
4 - 11	Channel name (eight characters).
12 - 15	Reserved.
16 - 17	File usage flags; bit 15, the channel descriptor flag, is set to one.

Byte	Description
18	Channel flags. These flags define the channel.
19	Installed ID of owner task.
20	Default resource type. The resource type of the channel as it appears to the requesting task. The significance of the contents of this byte depends on the resource type flag (as described in a subsequent paragraph).
21	Resource type flags.
22 – 23	Maximum length for messages that the channel transfers.
24 – 29	Reserved.
30 – 31	Record number of next CDR or ADR. The record number of the next record in the linked list of CDRs and ADRs. This field contains 0 when this CDR is the last record in the list.
32 – 33	Record number of FDR of the channel owner task program file.
34 – 143	Reserved.
144	User ID. The user ID of the user who created the IPC channel.
145 – 255	Reserved.

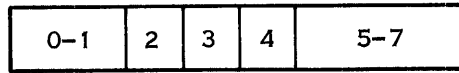
Of the file usage flags in bytes 16 and 17, bits 0 through 14 are reserved. Only bit 15, the CDR flag, applies. The flags have the following meanings:

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

2279397

Bit(s)	Meaning
0 – 14	Reserved.
15	CDR, as follows: 1 — Record is a CDR 0 — Record is not a CDR

The channel flags, byte 18, define the channel attributes as follows:



2279398

Bit(s)	Meaning
0 – 1	Scope of channel, as follows: 00 – Task local 01 – Job local 10 – Global
2	Shared, as follows: 1 – Channel is shared. 0 – Channel is not shared.
3	Symmetric, as follows: 1 – Symmetric channel 0 – Master/slave channel
4	Assign, as follows: 1 – Channel owner processes assign LUNO. 0 – Channel owner does not process assign LUNO.
5	Abort, as follows: 1 – Channel owner processes abort request. 0 – Channel owner does not process abort request.
6	I/O utility, as follows: 1 – Channel owner processes I/O utility request. 0 – Channel owner does not process I/O utility request.
7	Reserved.

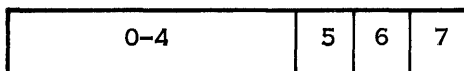
When the device resource type flag (bit 6, byte 21) is set, the default resource type (byte 20) has the following significance:

- 0 — Dummy device
- 1 — Special device
- 2 — 743 keyboard send/receive (KSR)
- 3 — 733 automatic send/receive (ASR)
- 4 — 733 cassette drive
- 5 — Reserved
- 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
- 19 — Reserved
- 20 — Bit-oriented/character-oriented asynchronous interface module (BCAIM)

When the file resource flag (bit 7, byte 21) is set, the default resource type (byte 20) has the following significance:

Value	Device
0	Reserved
1	Sequential file
2	Relative record file
4	Directory file
5	Program file
6	Image file

The resource type flags in byte 21 define the default resource type in byte 20. The flags are as follows:



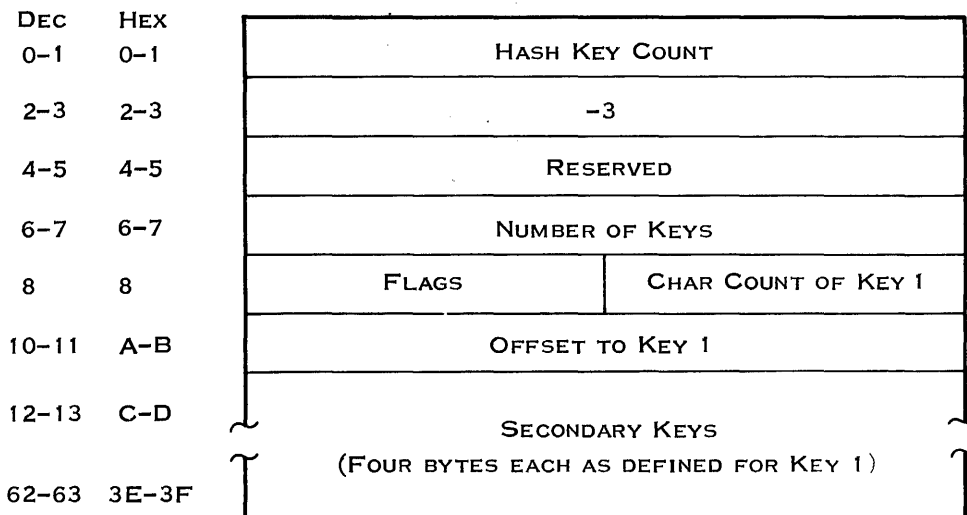
2279399

Bit(s)	Meaning
0 – 4	Reserved.
5	Channel resource flag, as follows: 1 — Default resource is an IPC channel. Byte 20 contains a channel resource type. 0 — Default resource is not an IPC channel.
6	Device resource flag, as follows: 1 — Default resource is a device. Byte 20 contains a device resource type. 0 — Default resource is not a device.
7	File resource flag, as follows: 1 — Default resource is a file. Byte 20 contains a file resource type. 0 — Default resource is not a file.

When the channel resource type flag (bit 5, byte 21) is set, the default resource type (byte 20) should contain 0 for a symmetric channel.

2.3.1.5 Key Descriptor Record (KDR). If the file being inserted is a KIF, the KDR requires another directory record. DNOS locates this record by searching linearly from the FDR of the file. The KDR is inserted in the first available directory record following the FDR.

A KIF has a primary key and can have as many as 13 secondary keys. A KDR describes the keys that access records in the file. Figure 2-9 shows the format of a KDR.



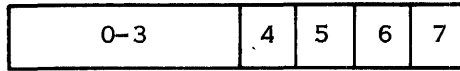
2279400

Figure 2-9. KDR Format

The KDR shown in Figure 2-9 contains the following information:

Byte	Description
0 - 1	Hash key count. The number of records in the directory that hashed to this record number. The KDR is not hashed. When this value is greater than 0, an FDR, ADR, or CDR has been written in the next available record because this record is occupied.
2 - 3	Hash key. This field corresponds to the hash key field of other directory records and contains -3, indicating that this is a KDR.
4 - 5	Reserved.
6 - 7	The number of keys defined for this KIF. A maximum of 14 keys are available for any KIF. One key, the primary key, is required. Keys 2 through 14 are optional secondary keys.
8	Primary key flags.
9	The key length, in bytes (characters), for the primary key.
10 - 11	The byte number of the first character of the key within the KIF data record.
12 - 63	Data for the secondary keys, if any. Four bytes in the format shown for the primary key (that is, key flags, key length, and key offset) are supplied for each secondary key.

The key flags are defined as follows:



2279401

Bit(s)	Meaning
0 – 3	Reserved.
4	Sequential placement flag. Applies only to primary key, as follows: 1 — Created by system using sequential placement scheme. 0 — Created by system using hash placement scheme.
5	Value present flag, as follows: 1 — Value need not always be present. (Valid only for secondary keys.) 0 — Value must always be present.
6	Sequential commands flag, as follows: 1 — Sequential commands are desired. 0 — Sequential commands are not desired.
7	Duplicates flag, as follows: 1 — Duplicate values are allowed for this key. 0 — Unique values are required for this key.

2.3.1.6 Example of a Dump Directory. Figure 2-10 shows a dump of the directory file .JB.DIR. The directory contains a sequential file (.JB.DIR.SEQ), an image file (.JB.DIR.IMAG), a program file (.JB.DIR.PROG), and a KIF (.JB.DIR.KEYFILE). The directory also contains an alias for the KIF (.JB.DIR.KEYFILE) and the KDR for the KIF. The directory was created to have seven entries in addition to record 0 (the directory overhead record).

2.3.2 Sequential Files

Sequential files support variable-length logical records. Logical records can span physical record boundaries regardless of ADU boundaries. When a logical record spans a physical record boundary, it is divided into partial records in separate physical records. The first word of each physical record has two flags, which indicate the following:

- Whether the first logical record is continued from the preceding physical record
- Whether the last logical record is continued to the following physical record

```

FILE ACCESS NAME:  .JB.DIR
RECORD:  000000
0000  0007 0004 0001 0000 4449 5220 2020 2020  .. .. .. .. DI R
0010  0002 4A42 2020 2020 2020 0300 0000 0000  .. JB      .. .. ..
    SAME
00FE  0000                                     ..
RECORD:  000001
0000  0000 0005 5345 5120 2020 2020 0000 0000  .. .. SE 0
0010  0A00 0300 0050 0001 068A 0001 0000 0000  .. .. .P .. .. .. ..
    SAME
0036  07BD 010E 9121 07BD 010E 91F1 0101 0000  .. .. .! .. .. .. ..
    SAME
0090  4A4F 4E20 2020 2020 0000 0000 0000 0000  JO N      .. .. .. ..
    SAME
00FE  0000                                     ..
RECORD:  000002
0000  0001 0002 5052 4F47 2020 2020 0000 0000  .. .. PR 0G      .. ..
0010  8C20 0120 0120 0011 4CAE 0001 0000 0000  .. . . . . L. .. .. ..
0020  0000 0033 0000 0033 0000 0000 0000 0000  .. .3 .. .3 .. .. .. ..
0030  0000 0000 0000 0000 07BD 010E 910B 07BD 010E  .. .. .. .. .. .. .. ..
0040  91EA 0103 0000 0000 0000 0000 0000 0000  .. .. .. .. .. .. .. ..
    SAME
0090  4A4F 4E20 2020 2020 0000 0000 0000 0000  JO N      .. .. .. ..
    SAME
00FE  0000                                     ..
RECORD:  000003
0000  0000 0000 0000 0000 0000 0000 0000 0000  .. .. .. .. .. .. .. ..
    SAME
00FE  0000                                     ..
RECORD:  000004
0000  0003 0004 4B45 5920 2020 2020 0000 0000  .. .. KE Y      .. ..
0010  1E08 0300 001C 002A 4CBF 0001 0000 0006  .. .. .. .* L. .. .. ..
0020  0000 0000 0000 0029 0001 0000 0029 0027  .. .. .. .) .. .. .) .
0030  0029 0059 0005 07BD 010E 9184 07BD 010E  .) .Y .. .. .. .. .. ..
0040  91EE 0101 001D 0000 0000 0000 0000 0000  .. .. .. .. .. .. .. ..
    SAME
0090  4A4F 4E20 2020 2020 0000 0000 0000 0000  JO N      .. .. .. ..
    SAME
00FE  0000                                     ..
RECORD:  000005
0000  0001 FFFD 0059 0002 080D 0000 0014 0009  .. .. .Y .. .. .. .. ..
0010  0000 0000 0000 0000 0000 0000 0000 0000  .. .. .. .. .. .. .. ..
    SAME
00FE  0000                                     ..
RECORD:  000006
0000  0000 0004 4B45 5946 494C 4520 0000 0000  .. .. KE YF IL E .. ..
0010  1E18 0000 0000 0000 0000 0000 0000 0000  .. .. .. .. .. .. .. ..
0020  0004 0000 0000 0000 0000 0000 0000 0000  .. .. .. .. .. .. .. ..
    SAME
00FE  0000                                     ..
RECORD:  000007
0000  0000 0004 494D 4147 2020 2020 0000 0000  .. .. IM AG      .. ..
0010  C420 0120 0120 0011 4CE9 0001 0000 0000  .. . . . . L. .. .. ..
    SAME
0036  07BD 010E 9118 07BD 010E 91F0 0103 0000  .. .. .. .. .. .. .. ..
    SAME
0090  4A4F 4E20 2020 2020 0000 0000 0000 0000  JO N      .. .. .. ..
    SAME
00FE  0000                                     ..

```

Figure 2-10. Dump of a Directory File

The flag bits, when set to 1, have the following meanings:

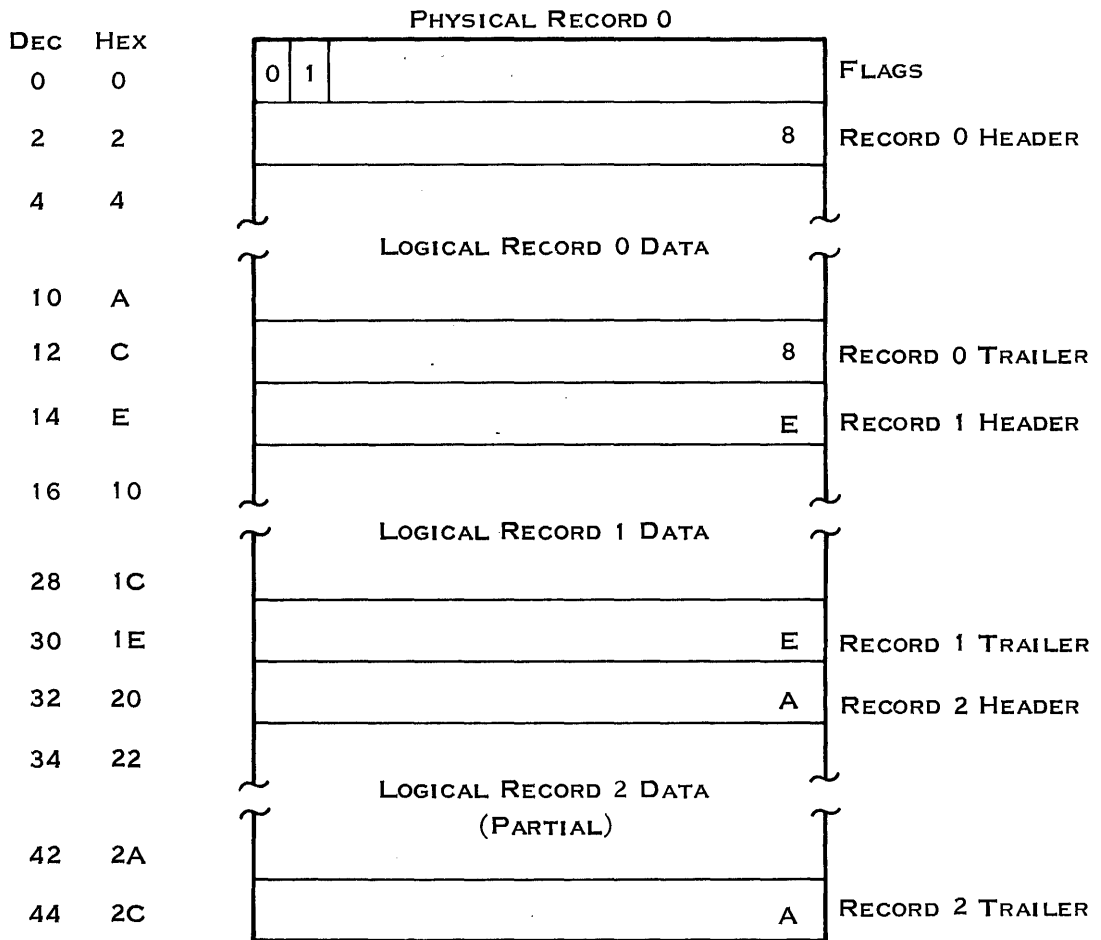
Bit	Meaning
0	First logical record in this physical record is continued from the preceding record.
1	Last logical record in this physical record continues in the next record.

Figure 2-11 shows the format of a sequential file. Each logical record or partial record is preceded by a header word and followed by a trailer word. The header and trailer words contain the number of bytes of user data. An EOF is signified by a zero-length record (zero header and trailer).

When a record ends with only one or two words remaining in the physical block, there is no room for another partial record (header/data/trailer). In this special case, the next record begins in the following block; the last word of the physical record is effectively a physical record trailer. It contains the number in the last trailer plus the number of unused bytes (two or four).

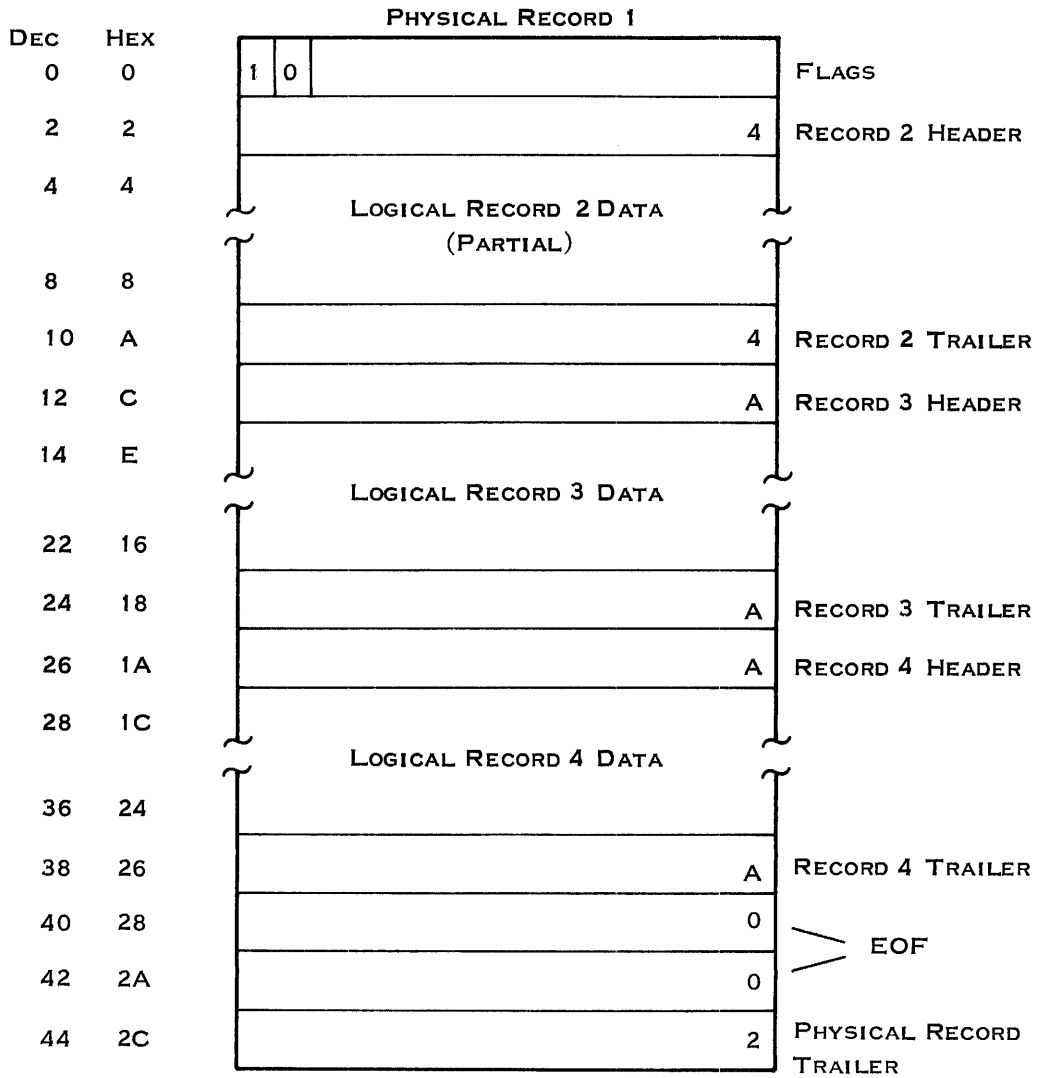
Logical records of a sequential file can be blank suppressed. (The sequential file is created blank suppressed.) In blank-suppressed files, words that contain two blanks are removed. A blank-suppressed logical record has the following format:

- Header word
- Byte containing a count of words of blanks
- Byte containing a count of words that contain at least one nonblank character
- Data characters
- Repetitions of items 2 through 4
- Trailer word



2279402 (1/2)

Figure 2-11. Sequential File Format (Sheet 1 of 2)



2279402 (2/2)

Figure 2-11. Sequential File Format (Sheet 2 of 2)

Figure 2-12 shows a blank-suppressed record. Notice that items 2 and 3 precede each group of characters (item 4) and that the number of words in item 3 is the length (in words) of item 4. In Figure 2-12, counts are hexadecimal, and hexadecimal ASCII representations are shown for characters.

INPUT RECORD:	0	0	1	1	2	2	3	3	•••	8
COLUMN:	1	5	0	5	0	5	0	5	•••	0
	FIRST			LAST			AGE		(COLUMNS 33-80 BLANK)	

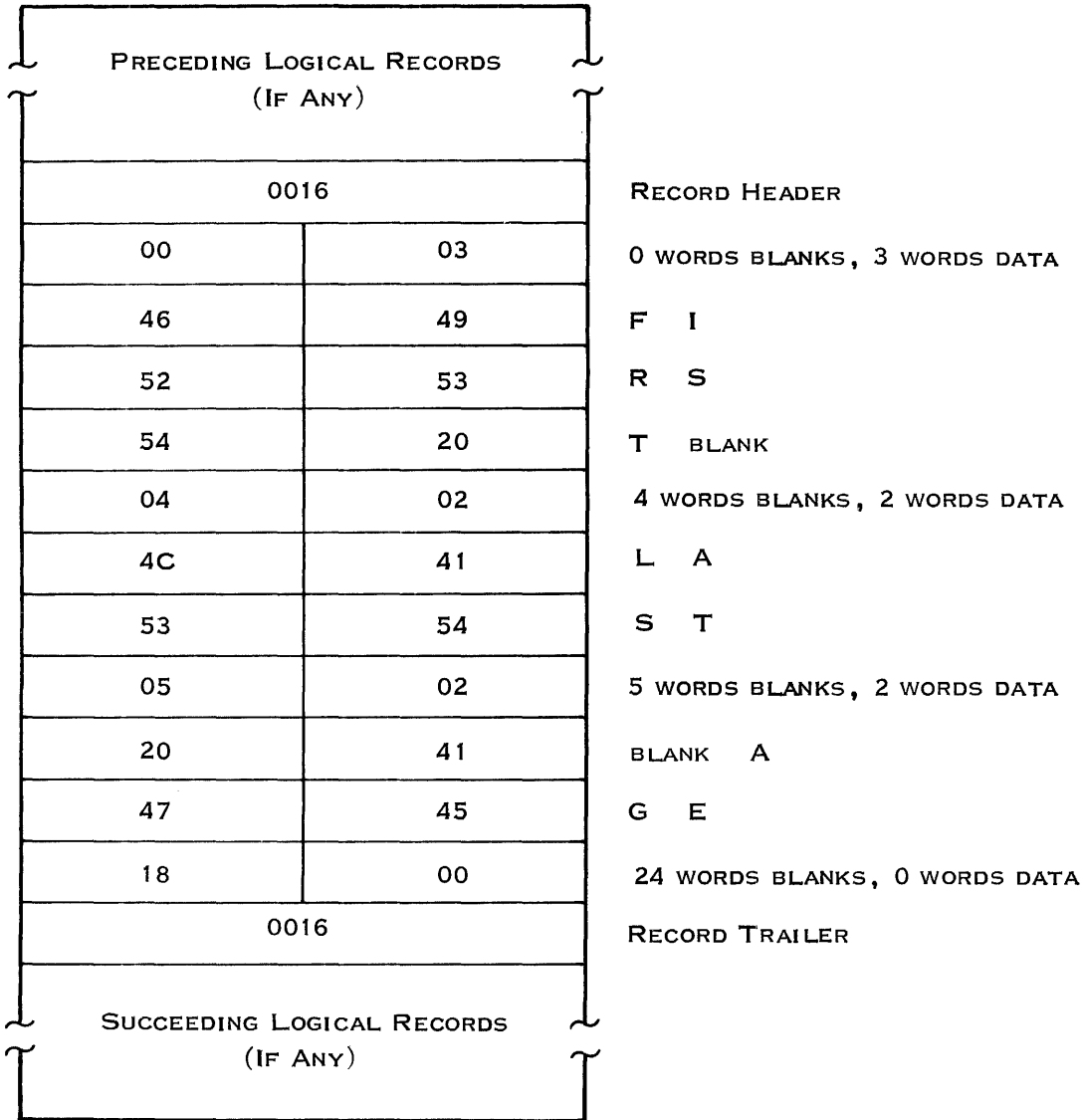
2279403

2.3.3 Relative Record Files

A relative record file is a file in which each logical record can be randomly accessed by its unique record number. All records in a relative record file are of the same length. Relative record files can be unblocked or blocked:

- Unblocked — The logical record size is greater than half of the physical record size.
- Blocked — The logical record size is less than or equal to half of the physical record size.

2.3.3.1 Unblocked Relative Record Files. Each logical record of a relative record file occupies one physical record of the file. A physical record should be any integral multiple of contiguous sectors. File accesses require reading or writing all sectors of a physical record. One disk access can read multiple contiguous sectors. Records read from unblocked relative record files are transferred directly from the disk to the user buffer without intermediate system buffering. When the user specifies a particular record of the file, the record number is converted to an absolute ADU number and a sector offset within the ADU. The absolute disk address is then passed to the disk device service routine (DSR) to perform the actual data transfer. The disk DSR converts the ADU and relative sector to the physical track and sector disk address that the disk controller hardware requires.



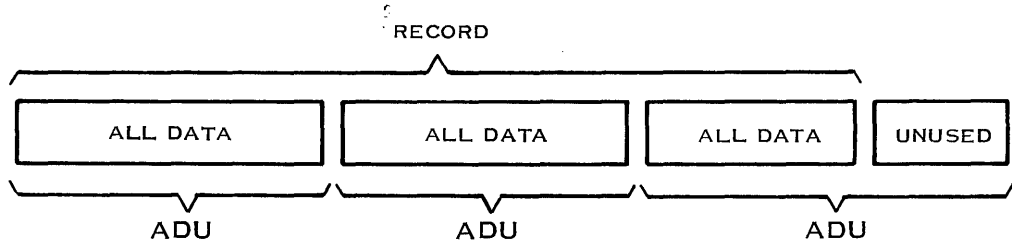
2279404

Figure 2-12. Blank-Suppressed Record

Each physical record must begin on a sector boundary. A physical record that begins on a sector boundary that is not also an ADU boundary cannot span the ADU boundary. The disk format for unblocked relative record files is as follows. In the first format, the record is larger than the ADU; in the second format, the record is smaller than the ADU.

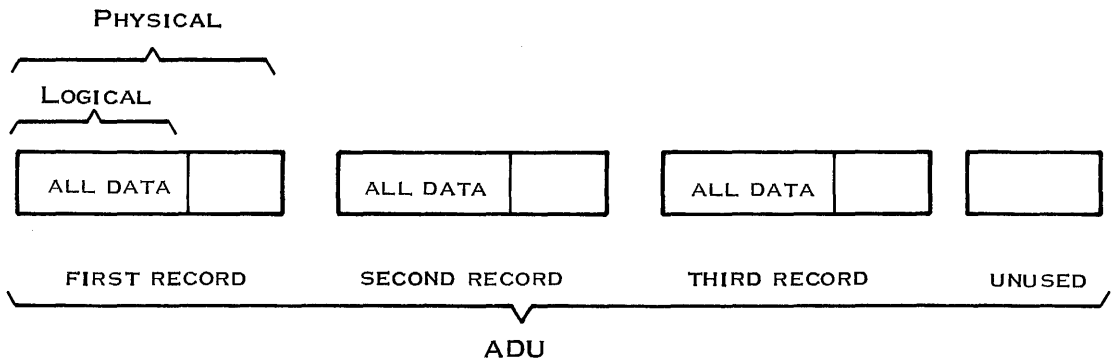
Unblocked Relative Record File

Record Size > ADU Size



2279405

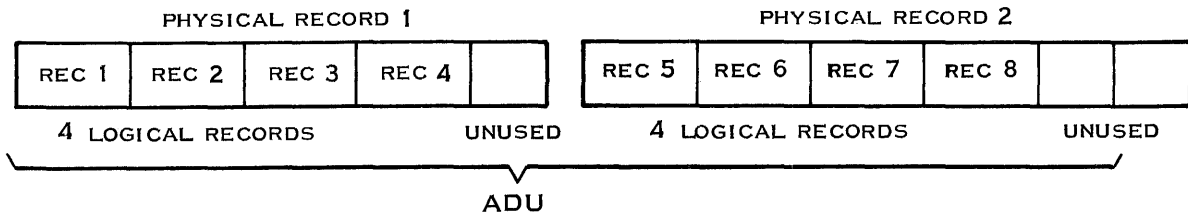
Record Size < ADU Size



2279406

2.3.3.2 Blocked Relative Record Files. These files are similar to unblocked relative record files except that multiple logical records can be stored in each physical record. Logical records cannot span physical records. Records are transferred via intermediate blocking buffers that are in the general pool of user space. The disk format for blocked relative record files is as follows:

Blocked Relative Record File



2279407

2.3.4 Key Indexed Files (KIFs)

A KIF is a file in which you can access records by the value of a character string called a key. Each KIF can have as many as 14 keys, with access through each key independent of the other keys.

Each entry of data made to the file is called a record. DNOS reads the records of other file types by identifying their positions in the file. In contrast, DNOS reads the records of KIFs by identifying a portion of the content of the record.

2.3.4.1 KIF Keys. A character field used to identify a record is called a key. A key is defined at the file level and applies to every record in the file. It is a static set of values that cannot be changed except by reconstructing the file. A KIF must have at least one key. KIF key fields are defined when the file is created and can be from 1 to 100 characters long.

The first key defined is the primary key; the others are defined as secondary keys. The primary key need not be in the first portion of a record; also, secondary key fields can physically precede the primary key within a record.

When you define a key, you can specify the following:

- Whether the key permits duplicates
- Whether the key is modifiable

If the value of a key must be unique throughout an entire file, the key must not permit duplicates. This prevents a record from being inserted into the file if the record has the same key value as a record already in the file. For example, keys such as employee numbers and social security numbers should not be duplicatable, while keys such as names and salaries should permit duplicates.

If a key is modifiable, you can change its value after a record with that key value has been inserted into the file. A key containing a person's salary should be modifiable, while a key containing the person's social security number should not be modifiable. If a record with a nonmodifiable key contains incorrect data, the only way to correct it is to delete and reenter the record. The primary key cannot be modifiable.

2.3.4.2 KIF Records. As you enter records into a KIF, they are logically sorted by key value. If more than one record has the same key value, they are sorted in the order they were entered.

The records of a KIF can be read either randomly or sequentially. When the records are read randomly, a key number and key value must be given for each Read operation. When the file is read sequentially, a key number and key value is supplied for the first read. The sorted order of that key determines the sequence of logical records returned by subsequent Read operations. The operation requests the next record either in forward or backward order.

2.3.4.3 KIF Key and Record Example. Since a key is defined for each record in the KIF, the records should contain related information, at least for the key portion. For example, related information could be a number (social security or employee number) or a name. The following example illustrates a KIF record and the keys the record contains:

1-9	10-20	21-30	31-40	41-46	47	48-52
123456789	DOE	JOHN	ANDREW	004442	M	02400

2279408

Key	Columns	Definitions
1	41 - 46	Employee number
2	01 - 09	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 - 47	Sex
8	48 - 52	Monthly salary

The record is 52 characters long and contains 7 fields. Since the name fields are used in more than one key, the record has 8 keys. Although this example does not include characters between keys, you have the option of entering blanks or other data between keys. Also, in the example every column is defined to be in at least one key. This is not required. Quite often only a small portion of the record is defined to be part of a key, while the rest of the record contains data. The only requirements are that a primary key must be included and that no key can be longer than 100 characters.

Since the primary key does not have to be in any particular position or have any qualities different from the other keys, you cannot determine which key is the primary key by looking at a KIF record. Instead, you can determine the primary key by entering a Map Key Indexed File (MKF) command. The primary key is identified as key number 1, as in the following example:

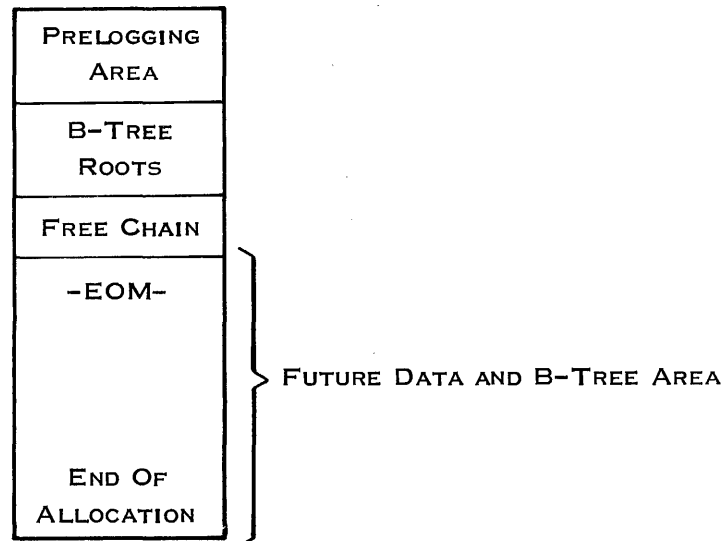
Key	Start Column	Length	Modifiable*	Duplicates Allowed*
1	41	6	N	N
2	1	9	N	N
3	10	11	Y	Y
4	21	10	Y	Y
5	31	10	Y	Y
6	10	31	Y	Y
7	47	1	N	Y
8	48	5	Y	Y

Note:

* Y indicates yes and N indicates no.

2.3.4.4 Structure of KIFs. The structure of a KIF consists of the prelogging area, the B-tree blocks, the data blocks, and the free chain blocks. A block is another term for a physical record. Figure 2-13 shows the structure of the KIF before any records have been inserted. The allocation for the file includes an area reserved for data and the B-tree area. The EOM is at the beginning of that area. As records are inserted, B-tree blocks and data blocks are added, moving the EOM toward the end of the allocation. The EOM indicates the current extent of the file. (KIFs are expandable.)

You can also compress these files. When you copy KIFs using the Copy Directory (CD) command or the Restore Directory (RD) command, you can compress the size to the EOM with the compress (CMP) option.



2279409

Figure 2-13. KIF Structure

Prelog Area

The first $(18 \times K) + 3$ physical records, where K is the number of keys defined for the file, are the KIF prelog blocks. Any physical record modified is first copied to a prelog block to prevent data loss in case of a fatal error during the data transfer. If a fatal error occurs, the logged image is written back into the original record on the next open of the file.

B-Tree

The next K physical records are the root nodes of the B-trees. Every defined key has a B-tree (up to 14 B-trees).

Free Chain

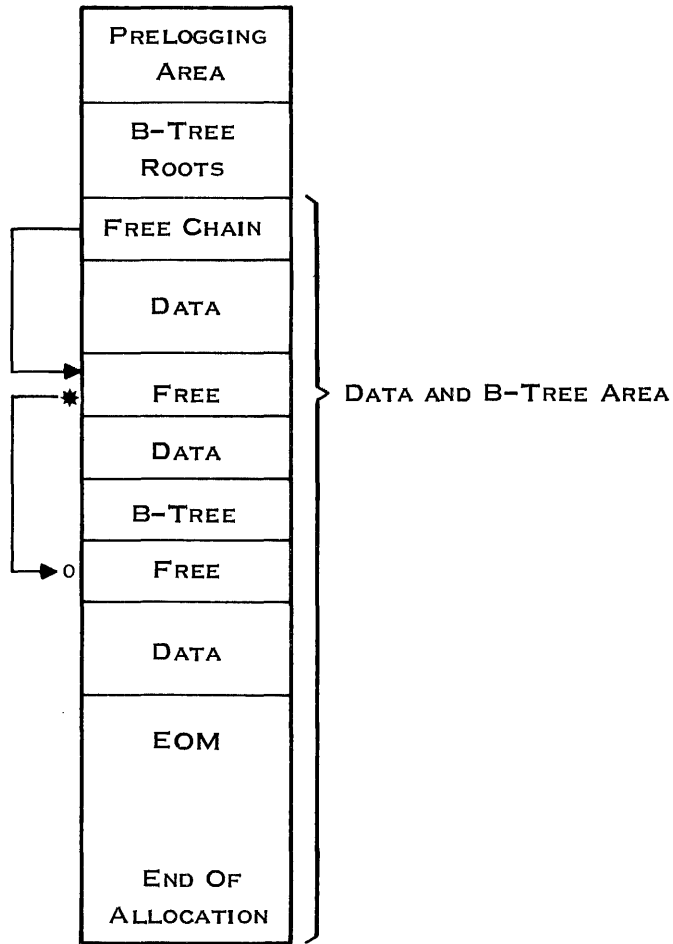
One block is created initially adjacent to the B-tree roots to contain the chain of free blocks. The block is accessed using a pointer in the file control block (FCB). The FCB is a memory-resident data structure of the file descriptor block.

When B-tree blocks become empty, the freed blocks are placed on the free chain. When a record is deleted from a data block, the data block is placed on the free chain. When a record is inserted into the file, it is placed in a block on the free chain.

Data Blocks

Data blocks contain the logical records of the file. All user data (logical records) is blank suppressed when stored in data blocks. The following paragraphs describe the structure of B-trees and data blocks.

Sequential Record Placement. When you insert data records into a KIF, the data records are placed in the data area sequentially. When you delete records from a file, available blocks are placed on the free chain to utilize available space. The KIF uses the available space on the free chain before using any space after the EOM. Figure 2-14 shows sequential record placement.

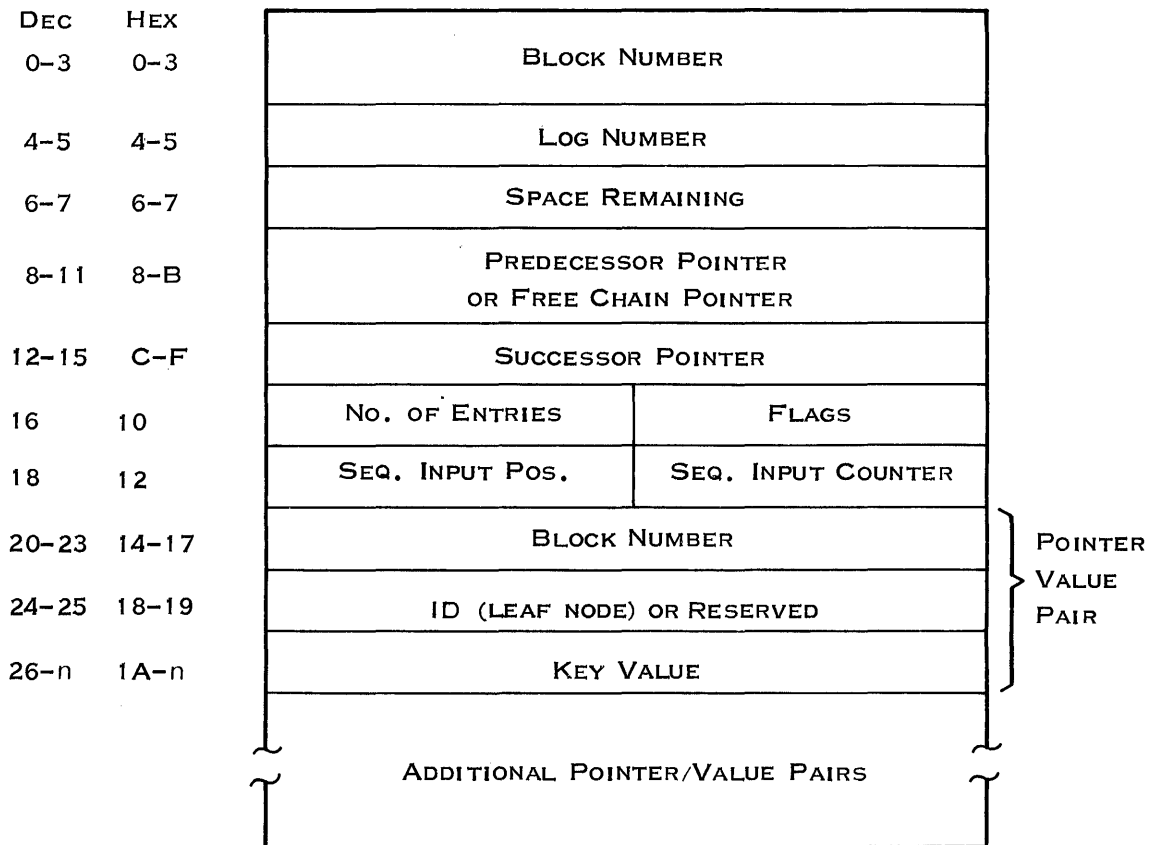


2279410

Figure 2-14. Sequential Record Placement Method

B-Trees. A B-tree is a balanced tree. It has multiple branches per node, and all leaf nodes are at the same level. DNOS B-trees can include as many as nine levels.

Each node of a B-tree occupies one physical record of a KIF and is called a B-tree block. The root node is initialized when the file is created, but all other nodes are created as records are added. Each B-tree block contains a few words of overhead and several pointer/key value pairs. Figure 2-15 shows the format of a B-tree block.



$$n = 26 (> 1A) + \text{LENGTH OF KEY}$$

2279411

Figure 2-15. B-Tree Block Format

The fields of the B-tree block are as follows:

Byte	Description
0 – 3	Physical record number of this B-tree block. Used for unlogging.
4 – 5	Log number that file management assigns when this block is logged.
6 – 7	Number of available bytes remaining in this B-tree block.
8 – 11	Preceding node on the same level; zero if leftmost.
12 – 15	Next node on the same level; zero if rightmost.
16	Number of pointer/value pairs in the B-tree block.
17	Flags, as follows: Bits 0 – 6 — Reserved Bit 7 — Set to 1 for leaf node; otherwise, set to zero
18	Sequential input position.
19	Sequential input counter.
20 – 23	Physical record number of the next lower node if this is not a leaf node. If it is a leaf node, physical record number of the data block containing the logical record associated with the key value.
24 – 25	For a leaf node, the ID of the logical record within the data block. Otherwise, this word is reserved.
26 – n	The actual key characters.

The remainder of the B-tree block contains more pointer/value pairs, each containing a physical record number and a key value (as in the pair that begins at byte 20). These entries in a B-tree block are kept sorted in increasing order of key value. The smallest key value is the first entry.

The log number in the B-tree block and in data blocks is a number that file management assigns when any operation that modifies any block in the file is performed. The same log number is placed in all blocks being modified during the operation. This is done as the blocks are logged (that is, copied into the prelog area). If you need to restore the file due to unsuccessful completion of the operation, the records in the prelog area with the log number of the operation are unlogged (copied back into the file).

The sequential input position, byte 18, and the sequential input counter, byte 19, require some explanation. When the B-tree block is created, byte 18 is set to 0 and byte 19 to the number of pointer/value pairs available plus 1. After the first insert, byte 18 is set to the number of keys in the block greater than the inserted key, and the value in byte 19 decreases by 1. Subsequent inserts decrease the value in byte 19 by 1 if the number of keys in the block greater than the inserted key equals the value in byte 18. If byte 19 equals 1 when the B-tree block is about to split, the ratio of the split will be 90/10 instead of 50/50. The 90/10 ratio indicates that the top 90 percent of the keys are placed in one block and the remaining 10 percent in another.

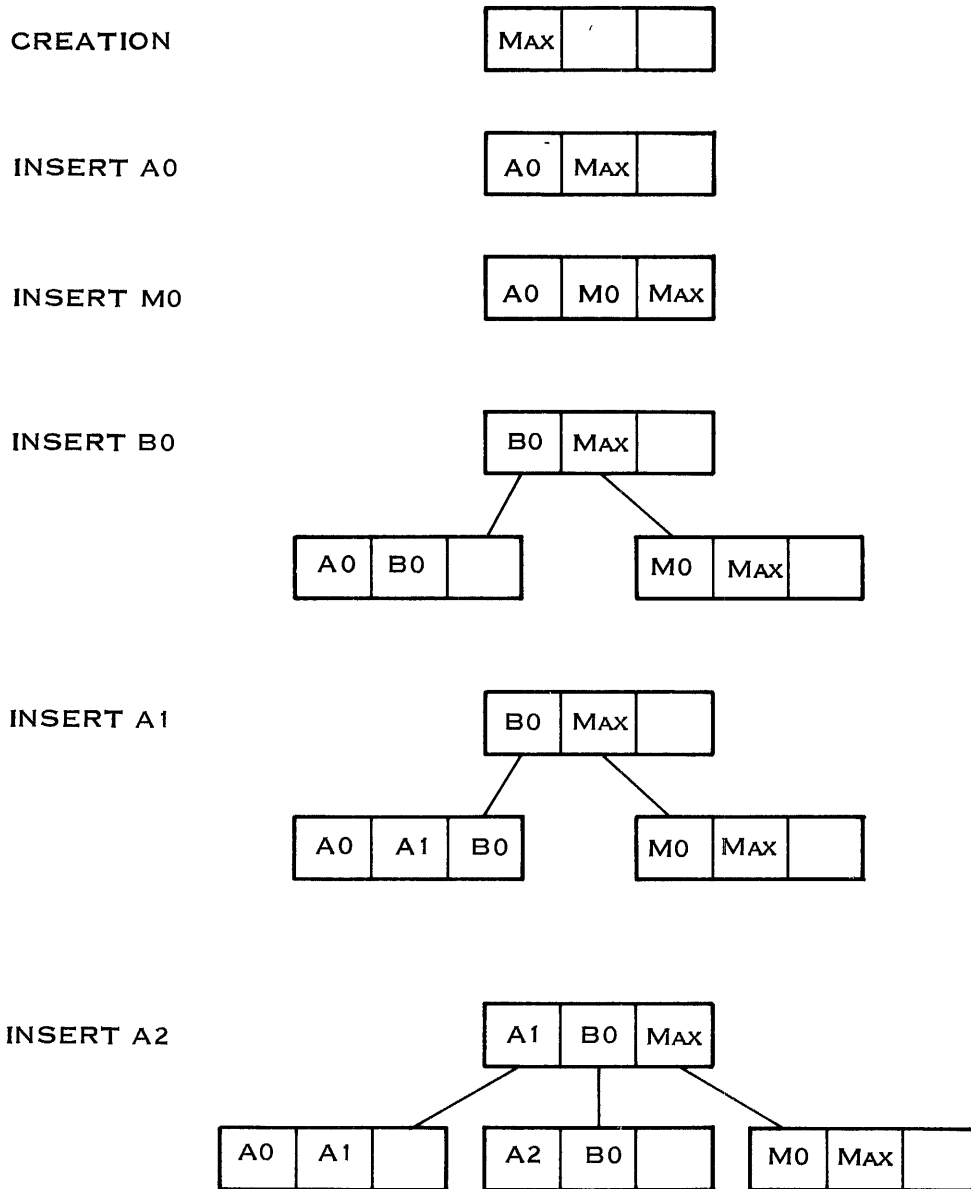
If the block is not a leaf node, each pointer field points to the root of the subtree that contains all key values less than or equal to the key value associated with the pointer. That is, the highest key value contained in the subtree is the key value associated with the pointer, as shown in Figure 2-16.

Figure 2-16 shows the development of the B-tree for a key of an example file. The key is two characters long, and each node has three pointer/value pairs. When the file is created, the root node contains one pointer/value pair, containing > FFFF, the maximum value of the key. The first operation inserts a record with a key of A0, resulting in two pointer/value pairs in the root node. Inserting another record, key M0, fills the root node.

The next record to be inserted has a key value of B0. The root node is split, producing a second level in the B-tree. For purposes of this example, all splits are 50/50. The new level contains two nodes, and the root node contains pointers to these nodes. The root node now contains keys B0 and > FFFF. The left node at the new level contains keys A0 and B0, and the right node contains keys M0 and > FFFF. Inserting a record with a key of A1 fills the left node at the bottom (leaf node) level.

When the record with a key of A2 is inserted, the left node splits, resulting in three leaf nodes. The nodes contain keys A0 and A1, A2 and B0, and M0 and > FFFF, from left to right. When the record with key A3 is inserted, it fills the center leaf node.

Inserting the record with a key of A4 again forces a new level. All nodes except the left and right leaf nodes are modified. The root node now contains keys A3 and > FFFF. The second level consists of two nodes. The left node contains keys A1 and A3, and the right node contains keys B0 and > FFFF. The new level contains four nodes with keys A0 and A1, A2 and A3, A4 and B0, and M0 and > FFFF, from left to right.

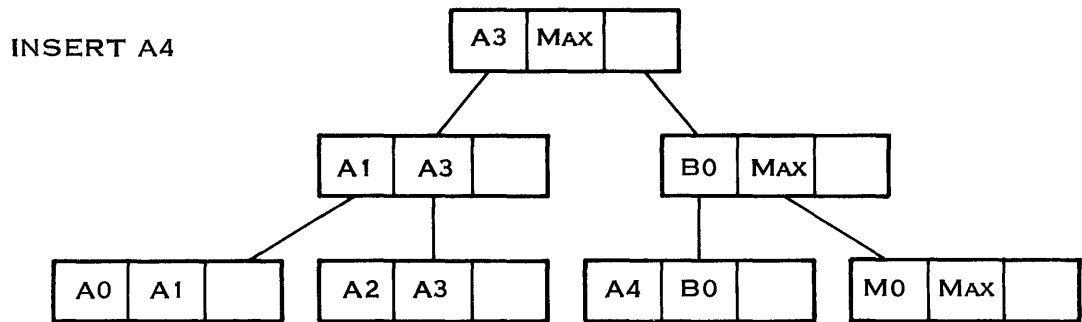
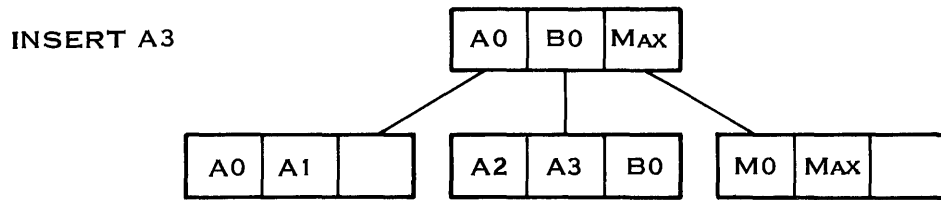


2279412 (1/2)

Note:

2-character keys; maximum is > FFFF; 3 keys per B-tree block

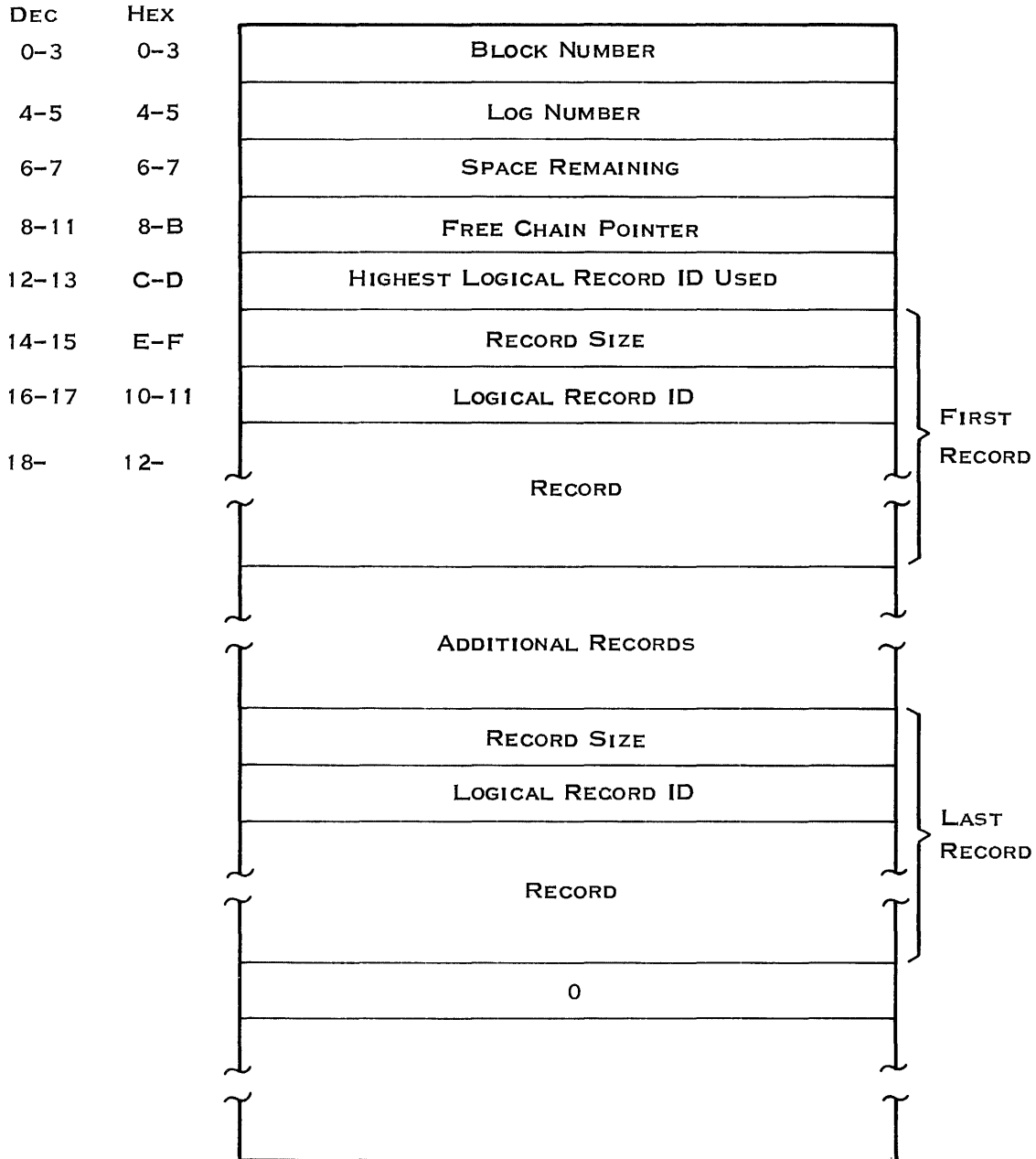
Figure 2-16. B-Tree Example (Sheet 1 of 2)



2279412 (2/2)

Figure 2-16. B-Tree Example (Sheet 2 of 2)

Data Blocks. A data block is a physical record of the file and contains a few words of overhead and several logical records as shown in Figure 2-17. The word following the last logical record has a zero value.



2279413

Figure 2-17. Data Block Format

The fields of a data block are as follows:

Byte	Description
0 – 3	Physical record number of the block.
4 – 5	Log number that file management assigns when this block is logged.
6 – 7	The number of bytes remaining in the block.
8 – 11	Free chain pointer. The block is placed in the free chain when a logical record is deleted from the block. The block may or may not include active logical records.
12 – 13	Highest ID assigned to any logical record within the data block.
14 – 15	Size (in bytes) of the first logical record inclusive.
16 – 17	The ID assigned to the first logical record.
18 –	First logical record.

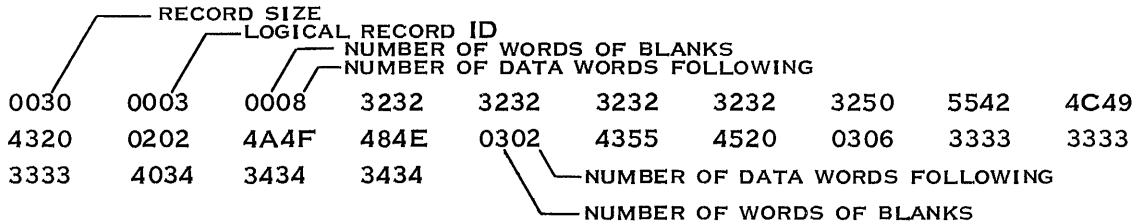
Additional records (if any) follow the first record. For each record, the size and ID precede the record. A word of zeros (the record size of the next record) follows the last logical record.

2.3.4.5 Description of Logical Record. A logical record in a KIF is a blank-suppressed record (described in paragraph 2.3.2). The first word of a blank-suppressed record contains the number of words of blanks removed and the number of words of data that follow. Following the specified words of data is another word with similar counts related to the next portion of the record. This pattern continues through the entire record, as shown in the following example:

1	10	21	31	41	47	48	52
22222222	PUBLIC	JOHN	CUE	333333	M	44444	

2279414

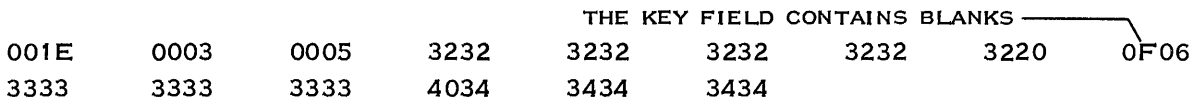
The record is written to the file as follows:



2279415

The records of a KIF that has only a primary key are a special case. The characters of the key are replaced by blanks in each record and are suppressed. The following example shows the record given in the preceding example with a primary key consisting of the entire name, columns 10 through 40:

SINGLE KEY ID = 3 KEY = ENTIRE NAME



2279416

The record contains 30 bytes instead of 48. The first word following the record number shows 5 words of data instead of 8. The word that precedes the last block of data replaces a field of 30 blanks.

2.3.4.6 KIF Disk Usage. This paragraph explains how to calculate the size of a KIF. The accuracy of the estimate depends on the accuracy of the parameters used in the calculation. These parameters are as follows:

- Physical record size
- Average blank-suppressed logical record size
- Sizes of all the keys
- Size of an ADU on the disk on which the file is created
- Maximum number of logical records
- Whether the input is sorted

The most difficult parameter to estimate is the average blank-suppressed logical record size. This is the average size of a logical record if all blanks are removed from all the records. You can easily determine the maximum number of logical records if the records are already in a sequential file. Otherwise, you must estimate this value. The other values are well defined and should require no estimations.

The disk allocation of a KIF consists of three specific areas:

- Prelogging area
- B-tree nodes
- Data records

The prelogging area is the only area of the three that has an absolute value. The following formula calculates the number of physical records of disk space required for this area:

$$\text{NPR}_{\text{prelog}} = (18 * K) + 3$$

where:

K is the number of keys.

NOTE

In the following formulas, [RD] means to round the number down to the nearest integer, and [RU] means to round the number up to the nearest integer.

The B-tree nodes are the records that contain the structures that make KIFs function differently from other file types. Only leaf nodes are included in this calculation so that the file estimate can be low by a few records. The following formulas estimate the number of physical records required for these structures:

$$X = \frac{\text{PRS} - 20}{\text{KS} + 6} [\text{RD}]$$

$$Y = \frac{\#LR}{X} [\text{RU}]$$

$$\text{NPR B-tree} = Y + (\text{SPLIT} * Y) [\text{RU}]$$

where:

PRS is the physical record size.

KS is the size of the key.

#LR is the maximum number of logical records.

SPLIT is 0.1 if the input is already sorted with respect to the key; otherwise, it equals 0.25.

You must determine a B-tree value for each key in the file.

The last area includes the data records. The following formulas estimate the number of physical records required for this area:

$$X = \frac{PRS - 16(RD)}{LRS + 6}$$

$$NPRdata = \frac{\#LR(RU)}{X}$$

where:

PRS is the physical record size.

LRS is the average blank-suppressed logical record size. (If the file has only one key, this value should not include the length of the key; that is, assume that the key consists of all blanks.)

#LR is the maximum number of logical records.

The 16 bytes subtracted from the PRS are the overhead of the physical data block. The 6 bytes added to the LRS are the overhead of each logical record.

The following formula calculates the total number of physical records required:

$$TPR = NPRprelog + NPRdata + \sum_{i=1}^K NPR \text{ B-tree } i$$

where:

K is the number of keys.

Finally, the following formula calculates the total number of ADUs required:

if $PRS \geq ADU$

then number of ADUs required = $\frac{PRS}{ADU} [RU] * NPR \text{ total}$

else number of ADUs required = $\frac{1}{\frac{ADU}{PRS} [RD]} * NPR \text{ total}$

where:

PRS is the physical record size.

ADU is the ADU size.

The following are examples of these calculations.

EXAMPLE 1

PRS = 864
 ADU = 864
 LRS = 60 (Average blank-compressed key size)
 KS = 20
 K = 1
 #LR = 800
 Sorted input (SPLIT = .1)

A) $NPR_{prelog} = (18 * 1) + 3 = 21$

B) $\frac{864 - 20}{32} [RD] = 32$

$20 + 6$

$\frac{800}{32} [RU] + (.1 * \frac{800}{32} [RU]) [RU] = 25 + 3$

$NPR \text{ B-tree} = 28$

C) $\frac{864 - 16}{16} [RD] = 12$

$60 + 6$

$NPR_{data} = \frac{800}{2} [RU] = 67$

$$\begin{aligned} \text{NPRtotal} &= \text{NPRprelog} + \text{NPR B-tree} + \text{NPRdata} \\ &= 21 + 28 + 67 \\ &= 116 \end{aligned}$$

$$\text{ADUs} = \frac{864}{864} [\text{RU}] * 116 = 116$$

EXAMPLE 2

PRS = 864
 ADU = 864
 LRS = 60
 KS1 = 20
 KS2 = 20
 KS3 = 20
 K = 3
 #LR = 2600
 Random input (SPLIT = .25)

A) $\text{NPRprelog} = (18 * 3) + 3 = 57$

B) $\frac{864 - 20 [\text{RD}]}{20 + 6} = 32$

$$\frac{2600 [\text{RU}]}{32} + (.25 * \frac{2600 [\text{RU}]}{32}) [\text{RU}] = 82 + 21$$

NPR B-tree(1) = 103 key 1
 NPR B-tree(2) = 103 key 2
 NPR B-tree(3) = 103 key 3

C) $\frac{864 - 16 [\text{RD}]}{60 + 6} = 12$

$$\text{NPRdata} = \frac{2600}{12} [\text{RU}] = 217$$

$$\begin{aligned} \text{NPRtotal} &= \text{NPRprelog} + \text{NPR B-tree(1)} + \text{NPR B-tree(2)} + \text{NPR B-tree(3)} + \text{NPRdata} \\ &= 57 + 103 + 103 + 103 + 217 \\ &= 583 \end{aligned}$$

$$\text{ADUs} = \frac{864}{864} [\text{RU}] * 583 = 583$$

Extending SCI

3.1 SCI OVERVIEW

The System Command Interpreter (SCI) is the principal interface between the operating system and the user. SCI operates in a job and executes commands in both the interactive and batch modes. Thus, SCI can execute in an interactive job at a terminal or from a batch stream without an associated terminal. Except for the way in which it accesses commands and their parameters, SCI executes in the same manner for the interactive mode as for the batch mode.

In the interactive mode, SCI displays prompts that request the values of command parameters. SCI can have one associated foreground task and one background task in the interactive mode.

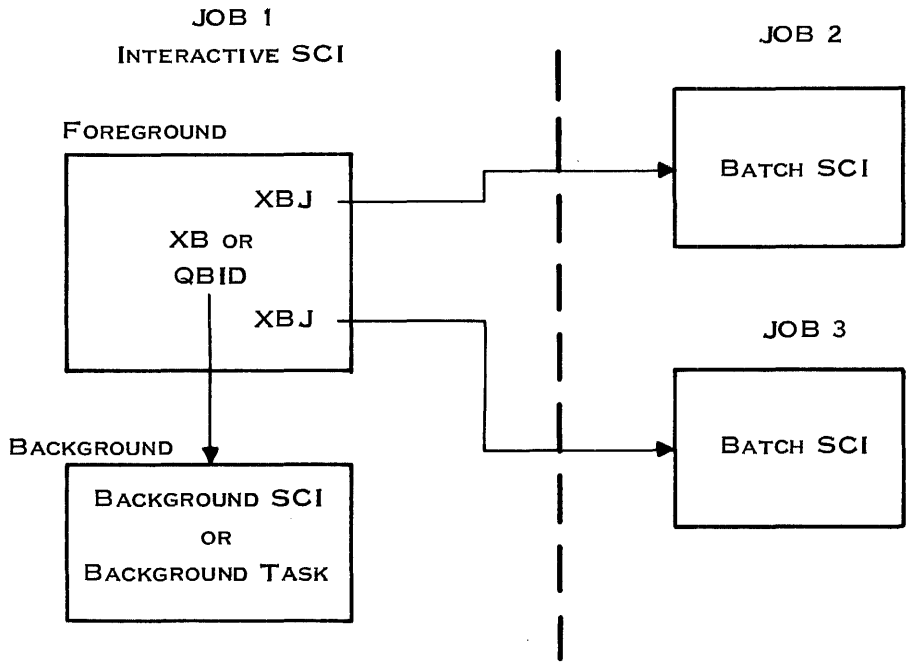
In the batch mode, you specify parameters by field prompt assignments in the command stream. The batch stream executes in background. A background task or an SCI batch job receives a copy of the synonyms and logical names when execution begins. Changes in the background or the batch mode synonyms and logical names do not affect the foreground values.

You can initiate an independent SCI batch job by using the Execute Batch Job (XBJ) command. Since batch jobs do not require associated terminals to execute, you can start any number of them from a terminal. Figure 3-1 illustrates the concept of SCI interactive and batch modes.

The name manager task maintains synonyms and logical names for jobs running under DNOS. Synonyms are local to a job; however, logical names are either job-local or global in scope. You can access synonyms and logical names for the job from SCI utilities or user tasks by issuing supervisor calls (SVCs) to the name manager. The name manager retrieves synonym and logical name definitions. However, you should call the appropriate SCI interface S\$ routines to access synonyms rather than issuing the SVC. This allows you to conform to any change in SCI implementation by linking the updated version of the S\$ routines.

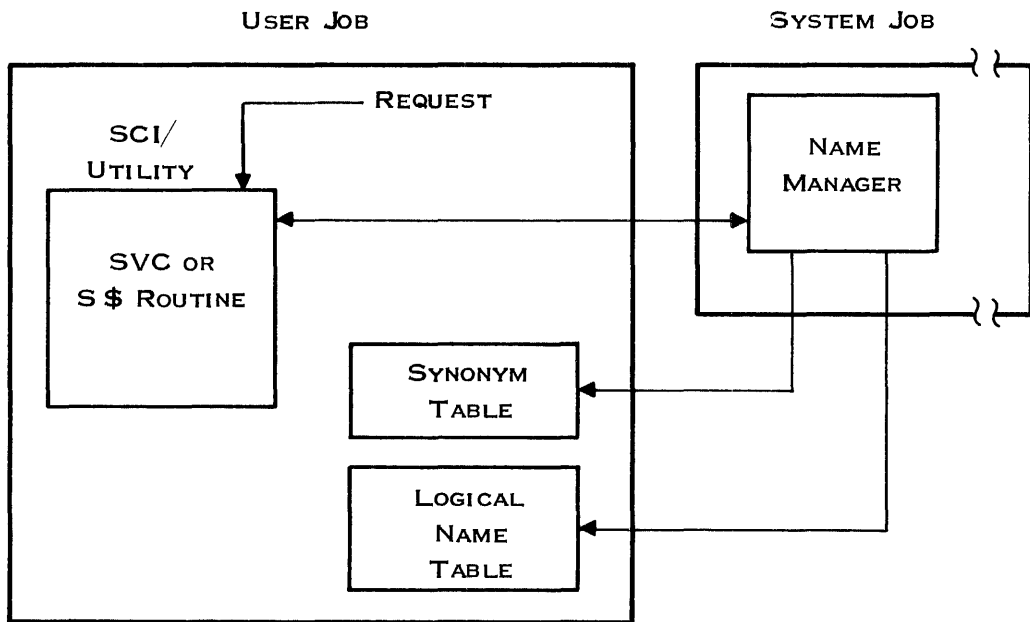
A synonym is a variable in the SCI language and represents either a string of characters or a null value. It functions as an alternative for another string. It is usually shorter than the text it replaces and more convenient to use.

A logical name is a user-specified character string used to name a resource within the scope of a job. A resource can be referenced by the logical name instead of a pathname or a device name. Consequently, a logical name resolves to a pathname or device name. A logical name can also appear as the first component of a pathname. Unlike synonyms, logical names can have associated parameters. This provides a general method of passing user-defined parameters to a task. Parameters are used to provide execution time values for SVC control blocks. Figure 3-2 shows the flow for accessing synonym and logical name tables from an SCI utility.



2279417

Figure 3-1. SCI Modes of Operation



2279418

Figure 3-2. SCI Access to Logical Names and Synonyms

The synonym table and logical name table are copied from a disk file when a user logs on. The file is usually identified with the user ID in the `.$USER` directory. However, it is also possible to use the Modify Terminal Status (MTS) command to have the user specify the file during logon. The name manager accesses memory copies of synonyms and logical names. The synonym and logical name table in memory are written back to the disk file when the interactive SCI terminates.

When several users are logged on with the same user ID and job name, they can share an environment of synonyms and logical names by responding YES to the RECONNECT prompt at log-on. Each user starts the session with the same set of synonyms and logical names. Each has his own environment as he makes changes. The environment of the last person to sign off from this job is saved on the disk.

In the interactive mode, SCI also uses the terminal local file (TLF). It provides a buffer on disk for lines to be displayed to the user. The lines are buffered so that the interactive SCI user can scroll through them. The name of the file is determined from the SCI mode and the terminal number, as follows:

- Foreground: `.$FTLFxx` (xx = terminal number)
- Background: `.$BTLFxx` (xx = terminal number)
- Batch SCI Job: `.$JLxxxx` (xxxx = job ID)

3.2 USER-DEFINED SCI COMMAND PROCEDURES

You can extend SCI by defining SCI command procedures for specific applications or by redefining or modifying the command procedures supplied with the system.

The next paragraphs give a brief overview of the following topics:

- SCI primitives
- Command procedures
- Command processors

After the overview, these topics are covered in detail as the rest of the section explains how to create your own SCI command procedures and command processors.

3.2.1 SCI Primitive

A primitive is the basic building block of the SCI language. Primitives allow you to create command procedures, which enable you to create additional commands that meet your application needs.

3.2.2 Command Procedure

A command procedure is a sequence of SCI statements (commands, primitives, or menu displays) that SCI executes. A command and its associated field prompts are defined in the command procedure.

You can use any existing SCI commands and any of the SCI primitives in a command procedure.

SCI command procedures are stored in a directory called a procedure library. Use separate libraries for the SCI commands provided with DNOS and for those you create. This precaution enables you to modify the libraries separately, since new releases can effect the SCI command library that comes with DNOS.

3.2.3 Command Processor

A command processor is a task that an SCI procedure executes to perform a specified action. The processor can be written in either a high-level language or assembly language. The processor can access synonyms and logical names to communicate with SCI. For many applications, the procedure calls the processor without passing any data to the processor. In more complicated cases, the procedure passes parameters to the processor via the PARMs parameter of .BID, .DBID, or .QBID.

The instructions or statements of the source code for the command processor vary with the language used and the action to be performed. The command processor can contain any number of instructions and can use the services of SCI.

3.3 SCI LANGUAGE SYNTAX

The SCI language consists of a set of commands (primitives), special characters, and variables. Command procedures use several SCI characters specifically defined for use in these procedures. Table 3-1 defines these special characters and the SCI language syntax, and subsequent paragraphs discuss primitives and variables.

Table 3-1. Special SCI Characters

Character	Meaning
!	Indicates the end of a record. Comments may occur after the ! but cannot span lines.
*	If it is in column 1, it indicates a comment statement; if it precedes a valid field prompt type (Table 4-4), it indicates that the field prompt is optional.
@	Indicates that SCI should treat the character string following the @ sign (and preceding the next nonalphanumeric character) as a synonym. If the synonym was previously defined, the value of the synonym replaces the @ sign and character string; otherwise, the value of the synonym is defined as the character string itself.
&	Indicates that the character string following the ampersand is a field prompt. The value replaces the ampersand and the character string; if no value is specified, a null string is indicated.
-	If it precedes a valid field type, the initial value and the user response are not echoed at the terminal. In a batch stream, the response is replaced by four dashes.
^	Delimits synonyms when concatenating them with other values or synonyms.
=	If it is in column 1, it causes the line in a batch stream to be executed but not written to the listing file.

In the following examples, lowercase characters indicate values supplied by the user; items enclosed by [] are optional.

The basic syntax of a command in the SCI language is:

```
command[field prompt(s)]
```

where:

command is the SCI command and field prompt(s) is a list of field prompt assignments.

At least one blank space must separate the command from the field prompt(s); blank spaces can also be entered before the command. Commas separate field prompts which can continue on successive lines.

The basic syntax of a primitive in the SCI language is:

```
primitive    [keyword list]
```

where:

primitive is the SCI primitive and keyword list is a list of keywords associated with the primitive.

At least one blank space must separate the primitive from the keyword list; blank spaces can also be entered before the primitive. The keywords are separated with commas and can continue on successive lines.

Any SCI language line (except a comment line) whose last nonblank character is an equals sign (=) or a comma (,) is continued on the following line.

The following example is processed as a single SCI command:

```
  IDT    YEAR=1980, MONTH=4, DAY=27,
        HOUR=18 ,MINUTE=56
```

Note that blank spaces can also follow or precede the commas separating field prompts. A continued list of field prompts can begin anywhere on subsequent lines. Field prompts on successive lines are usually placed directly below the initial line of field prompts for appearance and readability only.

3.4 SCI LANGUAGE VARIABLES

SCI language uses the following three types of variables:

- Synonyms
- Logical names
- Field prompts

Synonyms and logical names allow you to reference an I/O resource by an abbreviated name of the resource and are applicable to every task in a job. Field prompts are assigned values determined by the field prompts within a command. The following paragraphs define these variables and discuss their uses.

3.4.1 Synonyms

Synonyms are names that you assign to represent I/O resources using any of the following:

- .SYN primitive
- Assign Synonym (AS) command
- S\$SETS used in an application program
- Name Manager SVC (> 43) execution

The first three are user interfaces to the Name Manager SVC. This section discusses the .SYN primitive, S\$SETS, and the AS command. The *DNOS System Command Interpreter (SCI) Manual* explains the AS command and the *DNOS SVC Reference Manual* explains the Name Manager SVC execution.

The following example shows how the .SYN primitive defines a synonym equivalent to the directory component of a pathname:

```
.SYN      MY=DS02.MKC.SOURCE
```

The preceding example assigns the synonym MY to represent the directory DS02.MKC.SOURCE. As a result, a file in this directory can be referenced as follows:

```
[ ] SF
    PATHNAME: MY.DATA
```

When using synonyms, there are three ways to determine a synonym value, as follows:

- In the context of an SCI command procedure, precede the synonym name with an at sign (@).
- Use S\$MAPS/S\$SNCT in an application program.
- Execute the Name Manager SVC (> 43).

The first two are user interfaces to the Name Manager SVC. S\$MAPS and S\$SNCT are discussed later in this section.

When the @ sign precedes the synonym name in a string or expression, the synonym value replaces the synonym. For example, if the string DEVICE is previously defined as a synonym with the value LP01:

```
"LISTING DEVICE IS @DEVICE"
```

is evaluated by SCI as:

```
"LISTING DEVICE IS LP01"
```

If a synonym does not have a previously assigned value, SCI uses the synonym name as the value. For instance, if the synonym MY has not been defined, then the following is true:

```
.SYN X=@MY.MKC.DATA
```

is equivalent to

```
.SYN X=MY.MKC.DATA
```

SCI reads the synonym string characters from right to left, identifying the string following the @ sign as a synonym name. If there is more than one @ sign within a string of characters and the @ signs are not preceded by a special character (that is, the character is not a dollar sign, bracket, back slash, or alphanumeric), the string following the first @ sign encountered is evaluated and the synonym name is replaced by the value.

For example, if the following synonym is defined as:

```
.SYN GHI=123
```

and is used in the following synonym definition:

```
.SYN ABC=@DEF@GHI
```

SCI evaluates the string GHI, reading from right to left, and replaces the synonym with its defined value as follows:

```
.SYN ABC=@DEF123
```

SCI begins reading from right to left again, finds the @ sign and evaluates the entire string @DEF123 (not @DEF). Since DEF123 has not previously been defined, the .SYN primitive assigns the character string DEF123 as the value for the synonym ABC.

Assuming the GHI synonym is still defined as 123 and the synonym DEF is assigned the following value:

```
.SYN DEF=SRC
```

the following synonym definition:

```
.SYN ABC=@DEF .@GHI
```

would be evaluated as follows:

```
.SYN ABC=SRC.123
```

In situations where there is no special character in the string, use the caret (^) to separate two synonyms or enclose a synonym. The caret allows proper synonym evaluation by SCI, demonstrated in the following example.

The synonyms OBJ and PGM are defined and used in the following .SYN primitive:

```
.SYN OBJ=MY
.SYN PGM=PROGA

.SYN RESULT=@OBJ^@PGM
```

SCI evaluates the synonyms OBJ and PGM separately when the caret is inserted and defines the synonym RESULT as:

```
.SYN RESULT=MYPROGA
```

This process of reading and evaluating synonyms applies to all commands and is not unique to the .SYN primitive; the .SYN primitive is used only as an example for simplicity. When a line is read by SCI, textual substitution is performed immediately from right to left, without regard to the command being processed. After the synonym is evaluated, the command line is processed accordingly.

Because of the textual substitution on a line-by-line basis, a primitive split over several lines can have different results from a primitive written on one line. Consider the following example:

```
.SYN X=ABC
.SYN D=1
.SYN D=@X, E=@D
```

These lines generate the value ABC for D and the string 1 for E. Now, consider a second example:

```
.SYN X=ABC
.SYN D=1
.SYN D=@X,
      E=@D
```

These lines generate the value ABC for both D and E because the textual substitution in the last line occurs after the .SYN D = @X line is processed.

A synonym can be used without any problems to represent an entire pathname or only the first component of a pathname. However, because of the significance of special characters in the evaluation of synonyms, the use of synonyms to represent secondary components of a pathname can cause problems. For example, if S is a synonym defining SOURCE and is used in @VOL1.MYDISK.S, the synonym is not properly evaluated with the @ sign preceding the pathname. Synonyms can be used as secondary components if the @ signs are properly placed in the string evaluation. The correct synonym representation is VOL1.MYDISK.@S and is evaluated as VOL1.MYDISK.SOURCE.

The period acts as a delimiter in concatenation for synonyms. For example, if the synonym ABC has a value of XYZ, the concatenation of ABC to the character string .DEF would have the following results:

`@ABC.DEF=XYZ.DEF`

A character string can be concatenated to a synonym, as shown in the following example (where LAST represents the character string):

`LAST@XYZ`

where:

`@XYZ` represents the value of a synonym.

In addition to the synonyms you create, SCI maintains some synonyms which can be accessed by command procedures. These synonyms are listed in Table 3-2.

Table 3-2. SCI Maintained Synonyms

Synonym	Definition
\$\$BT	Task run ID of last background task.
\$\$BC	Completion code of last executed background task.
\$\$CL	List of current command procedure directories
\$\$MO	Two-digit hexadecimal code for the SCI mode: 00 = Batch mode 01 = TTY mode 0F = VDT mode
\$\$SI	Eight-character site name for this system.
\$\$ST	Two digit decimal station number (for example, 09). When executing in a batch job, \$\$ST is assigned a value of 00.
\$\$UI	User ID of one to eight characters (for example, SYSTEM).
ME	Four-character station name (for example, ST09). When executing in a batch job, ME is not assigned a value.

Table 3-3 lists the synonyms which are generated by command processors and SCI when using the error handling facility.

Table 3-3. Message Processing Synonyms

Synonym	Definition
\$\$CC	Hexadecimal completion code that can be returned by a command processor via the S\$TERM or S\$STOP routine
\$\$ES	Error source Indicator for the last status or error message
\$\$FN	File name within directory .S\$MSG from which the last message was generated
\$\$MN	Internal message number of the last error or status message declared by either SCI or a command processor
\$\$VT	Text string containing information about the last error or status message

3.4.2 Logical Names

Logical names appear functionally equivalent to synonyms, but they are significantly different. Like synonyms, logical names are sets of names and values; however, logical name values can include a set of parameters in addition to the resource name.

The logical name value is always assumed to be an I/O resource. Values associated with the logical names are descriptions of I/O resources, such as logically concatenated files or spooler devices. Logical names can have pathnames and descriptive parameters (for example, job-temporary or ANSI format) and can be job-local or global in scope. The resolution automatically occurs within DNOS each time the logical name is used in a context as an I/O resource. Treat the logical name as an I/O resource once it is defined.

There are two ways that you can define a logical name:

- Execute the Assign Logical Name (ALN) command.
- Execute the Name Manager SVC (> 43).

The ALN command is a user interface to the Name Manager SVC and is described in the *DNOS System Command Interpreter (SCI) Reference Manual*. Refer to the *DNOS SVC Reference Manual* for an explanation of the Name Manager SVC.

3.4.3 Environment and Scope of Name Definitions

Within a single job, each task has access to the set of synonyms and logical names that other tasks of the job have assigned. The effects that these name definitions have on an executing task can be thought of as the task's environment.

When a background task is started via a .QBID primitive, a new snapshot of the SCI environment is made and the new task executes in that environment. None of the synonym and logical name definitions changed by either task (subsequent to the bid) affect the environment of the other.

3.4.4 Field Prompts

A field prompt is the character string which requests a valid response to execute an SCI command. This character string contains a maximum of 28 characters, including embedded blanks.

Reference to the field prompt value can be made by preceding the field prompt with an ampersand (&). The following example refers to the value of the specified field prompt:

&INPUT PATHNAME OR LUNO

Field prompt values can include an assigned synonym. To reference a field prompt which contains a synonym as its value, the at sign (@) precedes the ampersand. The following example indicates that the synonym resolution is to be performed on the field prompt value:

@&INPUT FILE PATHNAME OR LUNO

Use the ampersand also when concatenating character strings, strings, and variables. For instance, the character string ABC is concatenated to the value of the field prompt FILE in the following example:

ABC&FILE

A field prompt can specify an appropriate response type for the acceptable values. When defining response types for field prompts, enclose the response type in parentheses to indicate that a list can be accepted as the value.

Table 3-4 lists the types of valid field prompts. The brackets [] indicate optional items and the parentheses () indicate an initial value for the prompt.

Table 3-4. Valid Field Prompt Types

Types	Items
ACNM	[(initial value)]
DEFAULT	(initial value)
ELEMENT	(response[= replacement],...)[(initial value)]
INT	[(initial value)]
LACNM	[(initial value)]
NAME	[(initial value)]
RANGE	(lower bound, upper bound)[(initial value)]
STRING	[(initial value)]
YESNO	[(initial value)]

The following features are common among field prompts:

- An asterisk (*) preceding the field prompt type indicates a response is optional and need not be supplied.
- If an initial value begins with a dollar sign (\$), then a null string is used as the initial value.
- The value of a field prompt can be a single value or a value list; however, DEFAULT can only be a single value. When defining a field prompt type in a command procedure, enclose the type in parentheses to allow a value list. Code a value list as a sequence of single items separated by commas when defining the response to a field prompt interactively. However, if a value list is entered in a batch stream, enclose the list in parentheses. If a list contains only one item, parentheses are not required.

For example, in a command procedure, FILE=ACNM declares the field prompt FILE and requires a value of the type ACNM. INPUT=(ACNM) indicates that INPUT can be a single value or a list of values which are of the ACNM type. When defining responses to these prompts interactively, FILE=DS04.LIST and INPUT=MY.MKC.PROGA,MY.MKC.PROGB are valid value assignments for FILE and INPUT, respectively.

- Each field prompt type can have a specified initial value. Enclose initial values which are lists in quotation marks (""). The DEFAULT type requires that an initial value be specified. Represent the null value for a field prompt, referred to as a null string, by a pair of quotation marks ("").
- A field prompt can be specified as having more than one possible field prompt type. For example, a response to a prompt can be a pathname or LUNO as used in this Execute Task (XT) command prompt:

```
PROGRAM FILE OR LUNO=ACNM/RANGE(0,OFF)
```

The preceding example is known as an alternate prompt type. Separate each field prompt type by a slash (/). The DEFAULT type cannot be specified as an alternate type.

The following paragraphs discuss each field prompt type and its format. The prompts used in the format examples are used for simplicity and may not be in their complete forms.

3.4.4.1 ACNM Field Prompt Type. The ACNM field prompt type allows a response that is a file name, channel name, or device name. The following is an example of the ACNM field prompt type:

```
FILE PATHNAME=*ACNM("@$$F$P")
```

In the previous example:

- The asterisk indicates that the response is optional.
- The response must be a single value.
- The at sign (@) preceding the set of characters, \$SF\$P, indicates the string represents a synonym.
- The parentheses around the set of characters, “@\$SF\$P”, indicate that SCI will use the value of the synonym as the initial value for the field prompt.
- The quotation marks around @\$SF\$P cause the entire value of the synonym to be shown. Without the quotation marks, invalid parameter messages can occur.

3.4.4.2 DEFAULT Field Prompt Type. The DEFAULT field prompt type assigns a default value to a field prompt. The DEFAULT type has the following three characteristics:

- Syntax is not checked.
- Field prompts of the DEFAULT type are not displayed in interactive mode but they can be explicitly assigned a value in batch mode or expert mode.
- The field prompt is assigned a default value only when the previous value was not assigned to the field prompt in batch mode or expert mode.

The following example illustrates the DEFAULT field prompt type:

```
DISPLAY=DEFAULT(Y)
```

The Y is a response previously assigned to the field prompt. It is only necessary to enter a response when you do not want the default.

3.4.4.3 ELEMENT Field Prompt Type. The ELEMENT field prompt type allows a list of acceptable responses to a field prompt to be specified. Using the near-equality algorithm discussed in the *System Command Interpreter (SCI) Reference Manual*, SCI attempts to match the response entered with each element in the list. If the response fails to match any item or matches more than one item, the type verification fails. Each item in the list can have a replacement value. Whenever a specific item in the list is matched, the value assigned to the field prompt is the replacement value and not your response. If the terminal is in default VDT mode (see the Modify Terminal Status (MTS) command in the *System Command Interpreter (SCI) Reference Manual*), the replacement value is echoed to the screen and replaces your response.

The following example illustrates the ELEMENT field prompt type:

```
ARE YOU SURE=ELEMENT(Y=YES,N)
```

In this example, the response must begin with a Y or N character. It is recommended that replacement values match the response in accordance with the near-equality algorithm.

If you respond with YOU BETCHA, the value of ARE YOU SURE is YES. However, if the response was NO WAY, the value of ARE YOU SURE is NO WAY.

3.4.4.4 INT Field Prompt Type. The INT field prompt type allows a response to be a 32-bit hexadecimal or decimal integer expression in the range of >80000000 through >7FFFFFFF (–2147483648 through 2147483647). The following example illustrates the INT field prompt type with the initial value enclosed in parentheses:

```
PARM1= INT(0)
```

In this example, the parentheses around the value zero indicate the initial value.

3.4.4.5 NAME Field Prompt Type. The NAME field prompt type allows a response to be a character string beginning with a dollar (\$) sign or a letter (A – Z). The remaining characters of the string can contain letters, numbers, \$, [,], or \. The following example illustrates the NAME field prompt type:

```
TASK NAME=*NAME
```

In this example, an asterisk preceding the field prompt type indicates that the response is optional.

3.4.4.6 RANGE Field Prompt Type. The RANGE field prompt type has the same function as the INT type; however, in addition, you can specify numeric upper and lower bounds. The following example illustrates the RANGE field prompt type:

```
LUN0=*RANGE(0,255)
```

In this example:

- The asterisk preceding the field prompt type *RANGE(0,255) indicates that the response is optional.
- The response must be in the range of 0 through 255.

3.4.4.7 STRING Field Prompt Type. A STRING field prompt type allows a response that is a string which does not contain quotation marks, exclamation marks, equals signs, parentheses, or commas.

The initial value specified for a STRING type can be enclosed by quotation marks, denoting it as a quoted string. A quoted string can contain quotation marks, exclamation marks, equals signs, parentheses, or commas within the enclosed string. However, you must be cautious when you use a string containing quotation marks, as they must always be used in pairs.

An error occurs if an unpaired quotation mark is used within the string, as in the next example:

```
"ENTER TO "RESUME OPERATION"
```

There should always be an even number of quotation marks in a quoted string, as in the next example:

```
"ENTER TO "RESUME OPERATION""
```

A pair of double quotation marks can also be used to represent a null string ("").

The following example illustrates the STRING field prompt type:

```
INPUT=*(STRING)("@XE$S")
```

The following statements are true:

- The parentheses around the field prompt type STRING indicate that the response can be a single value or a value list.
- The asterisk preceding the field prompt type (STRING) indicates that the response is optional.
- The at sign (@) preceding the characters XE\$S indicates that the value of the synonym is to be substituted for the string XE\$S.
- The parentheses around the set of characters "@XE\$S" indicate that the value of the synonym XE\$S is used as the initial value for the field prompt.
- The quotation marks enclosing the set of characters @XE\$S allow the value of the synonym XE\$S to be a list of values.

3.4.4.8 YESNO Field Prompt Type. The YESNO field prompt type allows a response that is an alphabetic character string beginning with a Y or an N character. The following example illustrates the YESNO field prompt type:

```
ARE YOU SURE?=YESNO
```

In this example, the response must begin with a Y or an N.

3.5 SCI PRIMITIVES

SCI primitives are the lowest-level members of the SCI language and are used to create command procedures and command processors. When applicable, primitives follow the guidelines discussed in the preceding paragraphs. Table 3-5 lists the SCI primitive notations and Table 3-6 lists the available primitives and their associated parameters. Subsequent paragraphs discuss each SCI primitive.

Table 3-5. SCI Primitive Notation

Notation	Meaning
Uppercase	Enter the item as shown.
Lowercase	Enter an item of this type.
No marks	The item is required.
[]	The item is optional.
Item...item	More than one item of this type can be used. Items are separated by commas.
Italics	Indicates the type of item required.
/	Indicates alternate items.

Table 3-6. SCI Primitives

Primitive Command	Parameters
.PROC	<i>name</i> [(<i>full name</i>)] [= <i>int</i>][, <i>field prompt list</i>]
.EOP	
.PROMPT	[(<i>full name</i>)] [= <i>int</i>][, <i>field prompt list</i>]
.SYN	<i>name</i> = "value"... <i>name</i> = "value"
.EVAL	[<i>mode</i>][= YES/NO,] <i>name</i> = <i>value</i>
.SPLIT	LIST = (<i>list</i>)[, FIRST = <i>name</i>][, REST = <i>name</i>] or LIST = " <i>string</i> "[, FIRST = <i>name</i>][, REST = <i>name</i>] [, CHARACTER = " <i>string</i> "][, POSITION = <i>int</i>][, STATUS = <i>name</i>]
.SVC	[\$ <i>name</i>]DATA/BYTE/TEXT = <i>value</i> (s)... [\$ <i>name</i>]DATA/BYTE/TEXT = <i>value</i> (s)
.IF	<i>op1</i> , <i>relation</i> , <i>op2</i>
.ELSE	
ENDIF	
.LOOP	
.WHILE	<i>op1</i> , <i>relation</i> , <i>op2</i>

Table 3-6. SCI Primitives (Continued)

Primitive Command	Parameters
.REPEAT	
.UNTIL	<i>op1,relation,op2</i>
.EXIT	
.BID	TASK = <i>namelint</i> [,LUNO = <i>int</i>][,CODE = <i>int</i>] [,PROGRAM FILE = <i>acnm</i>][,PARMS = (<i>string...string</i>) [,UTILITY]
.DBID	TASK = <i>namelint</i> [,LUNO = <i>int</i>][,CODE = <i>int</i>] [,PROGRAM FILE = <i>acnm</i>][,PARMS = (<i>string...string</i>) [,UTILITY]
.QBID	TASK = <i>namelint</i> [,LUNO = <i>int</i>][,CODE = <i>int</i>] [,PROGRAM FILE = <i>acnm</i>][,PARMS = (<i>string...string</i>) [,UTILITY]
.DATA	[<i>acnm</i>][,EXTEND[= YES/NO]][,SUBSTITUTION[= YES/NO]] [,REPLACE[= YES/NO]]
.EOD	
.STOP	[TEXT = <i>string</i>][,CODE = <i>int</i>]
.USE	[<i>pathname...pathname</i>]
.OPTION	[PROMPT[= <i>string</i>]][,MENU[= <i>name</i>]] [,PRIMITIVES[= YES/NO]][,LOWERCASE[= YES/NO]]
.MENU	[<i>menu name</i>]
.SHOW	<i>filename...filename</i>

3.5.1 .PROC and .EOP Primitives

You can use the .PROC primitive to begin an SCI procedure definition which must end with the .EOP primitive. Use the .PROC primitive to install the command procedure into a command procedure library. The following represents the .PROC format:

```
.PROC name[(full name)] [= int][,field prompt list]
```

The *name* parameter, which must be the first parameter, defines the name of the procedure. You can give an optional *full name*, enclosed in parentheses, immediately following the name. The *full name* is displayed on the terminal when the procedure is executed.

The *int* field is optional and determines the privilege level used for the procedure being defined. This field must follow the *full name* if you specified a *full name*. Table 3-7 shows the different privileges which can be specified.

Command privilege levels are assigned according to your system knowledge and job requirements. If a user ID privilege level is numerically lower than the privilege level assigned to a particular command, you cannot issue that command. The system manager uses the Assign User ID (AUI) or the Modify User ID (MUI) command to establish privilege levels. Privilege levels can be assigned with respect to the power of the command and the knowledge and trustworthiness of the user. The default value for the privilege level is 0.

Table 3-7. Command Privilege Levels

Level	Meaning
0	Lowest level of access privilege; for example, Create File (CF)
1	Defined by the System Manager
2	System access level; for example, Kill Task (KT)
3	Defined by the System Manager
4	Management access level; for example, Assign User ID (AUI)
5	Defined by the System Manager
6	Combination of System and Management; for example, Execute System Generation Utility (XSGU)
7	Defined by the System Manager

The *field prompt list* is a character string following the optional privilege level. A *field prompt list* is formatted as:

field prompt = field prompt type

where:

the field prompt type is one of those listed in Table 3-4 and follows the rules defined for it.

A maximum of 22 field prompts can be defined for a command.

The following example illustrates the .PROC and .EOP primitives used in a command procedure:

```
.PROC  EX (EXAMPLE PROC)=0,
      INPUT PATHNAME=ACNM
      SF    FILE="'&INPUT PATHNAME"
.EOP
```

In this example, the *full name* of the command procedure is EXAMPLE PROC and the SCI command is EX. The specified privilege level is zero, therefore, the command is available to any user. INPUT PATHNAME is a field prompt. INPUT PATHNAME requires an ACNM field prompt type for its response. The SF command uses the response to INPUT PATHNAME as an initial value. This command procedure could be used to install the EX command on the system.

When you issue the EX command, EXAMPLE PROC is displayed along with the INPUT PATHNAME prompt. The cursor is positioned in the response field and is ready for your entry. Press the RETURN key after you enter a response. The SF command is bid by the EX command procedure and the file identified for the INPUT PATHNAME response is displayed. (It is not necessary to completely understand the command procedure at this point. Details of command procedures are explained later in this section.)

It is good programming practice to indent the command procedure to show the control structures in it. The .PROC processor preserves such indentation when it creates the output file.

3.5.2 .IF, .ELSE, .ENDIF Primitives

The SCI language uses the constructions IF-THEN and IF-THEN-ELSE to create a conditional primitive. The .IF primitive must be used in conjunction with the .ENDIF primitive. The .ELSE primitive is an optional primitive used with the .IF and .ENDIF primitives. The .ENDIF primitive terminates the .IF primitive.

```
.IF    op1, relation, op2
      .
      .
      .
.ELSE
      .
      .
      .
.ENDIF
```

If the .IF condition is true, then statements immediately following the .IF primitive are executed. If the condition is false, any statements following the .ELSE primitive (if present) are executed. Execution then continues with statements following the .ENDIF primitive.

The .IF primitive must contain a condition using a relation defined in the following list:

Relation	Meaning
<i>op1</i> ,EQ, <i>op2</i>	<i>op1</i> is equal to <i>op2</i>
<i>op1</i> ,NE, <i>op2</i>	<i>op1</i> is not equal to <i>op2</i>
<i>op1</i> ,GT, <i>op2</i>	<i>op1</i> is greater than (follows) <i>op2</i> in the ASCII collating sequence
<i>op1</i> ,LT, <i>op2</i>	<i>op1</i> is less than (precedes) <i>op2</i> in the ASCII collating sequence
<i>op1</i> ,GE, <i>op2</i>	either GT or EQ is true for <i>op1</i> and <i>op2</i>
<i>op1</i> ,LE, <i>op2</i>	either LT or EQ is true for <i>op1</i> and <i>op2</i>
<i>op1</i> ,IS, <i>op2</i>	<i>op1</i> is of type <i>op2</i>
<i>op1</i> ,ISNOT, <i>op2</i>	<i>op1</i> is not of type <i>op2</i>

The EQ, NE, GT, LT, GE, and LE relations allow *op1* and *op2* to be strings, variables, or concatenated strings. The *relation* parameter designates the type of comparison which is performed on the operands. If both *op1* and *op2* are numeric, a numeric comparison is made; otherwise, a string comparison is performed.

The IS and ISNOT relations require *op2* to be a field prompt type; alternate types cannot be specified. A check is made to verify that *op1* satisfies the type specified by *op2*. The following example illustrates the IS relation:

```

      .
      .
      .
STATION ID = RANGE(0,OFF)/ELEMENT(ME)("@$$ST")
      .
      .
      .
      .IF "&STATION ID", IS, RANGE(0,OFF)
      .SYN $XT$SID="&STATION ID"
      .ELSE
      .SYN $XT$SID="@$$ST"
      .ENDIF

```

Note that the field prompt defines alternate types of responses, RANGE and ELEMENT. The .IF statement verifies that the value specified was of the RANGE type.

You can use any SCI primitives (excluding .PROC and .EOP primitives) between the .IF and .ENDIF primitives. You can use the .IF primitive within other .IF primitives with a maximum of 32 levels of nested conditionals. The following example uses the .IF, .ELSE, and .ENDIF primitives; IAN and OAN of the CC (Copy Concatenate) command represent the input and output pathnames, respectively.

```
.PROC  EX(EXAMPLE PROC)=0,
      INPUT  PATHNAME=ACNM,
      OUTPUT PATHNAME=ACNM,
      DISPLAY OR COPY?=ELEMENT(D=D,C=C)(DISPLAY)
.IF    &DISPLAY,EQ,D
      SF FILE="&INPUT PATHNAME"
.ELSE
      CC IAN="&INPUT PATHNAME",
      OAN="&OUTPUT PATHNAME"
.ENDIF
.EOP
```

The .IF primitive compares the &DISPLAY to the character D. &DISPLAY is the value of the response to the DISPLAY OR COPY? prompt; the character D represents a possible value for the response. If &DISPLAY and D are equivalent, the SF command procedure is bid and the value represented by &INPUT PATHNAME is displayed. If the response to DISPLAY OR COPY? does not match D, the .ELSE primitive bids the Copy/Concatenate (CC) command procedure to copy the contents of the file specified for INPUT PATHNAME to the file specified for OUTPUT PATHNAME. The .ENDIF primitive terminates the .IF comparison and execution continues with the primitives or commands following the .ENDIF.

3.5.3 .PROMPT Primitive

The .PROMPT primitive reduces the need for secondary command procedures, avoiding large command procedure libraries. Additional overhead involved in processing a new command procedure is also eliminated. The syntax for the .PROMPT primitive is as follows:

```
.PROMPT [(full name)][ = int][,field prompt list]
```

The field prompts defined by .PROMPT are displayed on a different screen from those defined by .PROC (that is, the screen is cleared and the new prompts are displayed).

The *full name* parameter is optional and specifies a character string to be displayed when the primitive is executed in interactive mode. The *int* parameter is the lowest privilege level assigned to the user ID which permits execution of the procedure. The privilege level specified for .PROMPT can be higher than the level specified by .PROC; however, it is not recommended. The *field prompt list* parameter is a series of field prompts.

The following example illustrates the .PROMPT primitive:

```
.PROC  EX(EXAMPLE PROC)=0,
      INPUT PATHNAME=ACNM,
      OUTPUT PATHNAME=ACNM,
      DISPLAY OR COPY?=ELEMENT(D=D,C=C)(DISPLAY)
.IF    &DISPLAY,EQ,D
      SF FILE("&INPUT PATHNAME"
.ELSE
      CC IAN("&INPUT PATHNAME",
      OAN("&OUTPUT PATHNAME"
.PROMPT (SUPPLEMENTARY QUESTION),
      DELETE FILE?=ELEMENT(Y=Y,N=N)(NO)
.IF    &DF,EQ,Y
      DF PATHNAME("&INPUT PATHNAME"
.ENDIF
.ENDIF
.EOP
```

The .PROC, .IF, .ELSE, and .ENDIF primitives execute as previously explained. The .PROMPT primitive displays the DELETE FILE? prompt. The character string SUPPLEMENTARY QUESTION defines the optional *full name* parameter.

3.5.4 .SYN Primitive

The .SYN primitive assigns values to synonyms. Synonyms and their values are maintained in the synonym table. The .SYN primitive has the following format:

```
.SYN name = "value"...name = "value"
```

The *name* parameter specifies the synonym name. The *value* parameter is a string, variable, or concatenated expression. Quotation marks enclosing the value are recommended to ensure correct interpretation of the value string.

A string enclosed in quotation marks indicates that the value specified is a list and is to be treated as a single item. A value list must be enclosed in quotation marks as shown in the following example:

```
.SYN A = "1,2,3"    Legal
```

```
.SYN A = 1,2,3     Illegal
```

Synonyms can be assigned string values containing special characters. To properly handle the special character, enclose the string in quotation marks. For instance, as a special character, the exclamation mark (!) indicates an end of record. If you use the ! in a character string, enclose the string in quotes to include the characters following it. For example:

```
.SYN A = "HELLO!THERE"
```

THERE is included in the string.

```
.SYN A = HELLO!THERE
```

THERE is not included in the string and is regarded as a comment.

To avoid synonym table overflow, delete synonyms which are no longer necessary. Assigning a null value ("") to a specified synonym deletes the synonym from the synonym table.

The following example illustrates the .SYN primitive used in a command procedure:

```
.PROC  EX(EXAMPLE PROC)=0,
      INPUT  PATHNAME=ACNM,
      OUTPUT PATHNAME=ACNM,
      DISPLAY OR COPY?=ELEMENT(D=D,C=C)(DISPLAY)
.SYN   $EX$IP("&INPUT PATHNAME")
.SYN   $EX$OP("&OUTPUT PATHNAME")
.IF    &DISPLAY,EQ,D
      SF FILE=@$EX$IP
.ELSE
      CC IAN=@$EX$IP,
      OAN=@$EX$OP
.PROMPT (SUPPLEMENTARY QUESTION),
      DELETE FILE?=ELEMENT(Y=Y,N=N)(NO)
.IF    &DF,EQ,Y
      DF PATHNAME=@$EX$IP
.ENDIF
.ENDIF
.EOP
```

This example is similar to the example for the .PROMPT primitive. In addition, this command procedure assigns values to the synonyms \$EX\$IP and \$EX\$OP which you can access by other command procedures.

3.5.5 .EXIT Primitive

The .EXIT primitive terminates the execution of the current command procedure. (Use the .EOP to terminate the definition of the command procedure.) The .EXIT primitive can be used as often as necessary at any point within a command procedure definition. Do not use the .EXIT primitive however, within a batch stream.

The .EXIT primitive does not have any parameters to be defined, and it uses the following format:

```
.EXIT
```

An example of the .EXIT primitive used in a command procedure is as follows:

```
.PROC EX(EXAMPLE PROC)=0,
    INPUT PATHNAME=ACNM("@$EX$IP"),
    OUTPUT PATHNAME=ACNM("@$EX$OP"),
    DISPLAY OR COPY?=ELEMENT(D=D,C=C)(DISPLAY)
.SYN  $EX$IP("&INPUT PATHNAME"
.SYN  $EX$OP("&OUTPUT PATHNAME"
.IF   &DISPLAY,EQ,D
    SF FILE=@$EX$IP
.EXIT
.ENDIF
    CC IAN=@$EX$IP,OAN=@$EX$OP
.PROMPT (SUPPLEMENTARY QUESTION),
    DELETE FILE?=ELEMENT(Y=Y,N=N)(NO)
.IF   &DF,EQ,Y
    DF PATHNAME=@$EX$IP
.ENDIF
.EOP
```

If the following statement from this example is true:

```
.IF &DISPLAY,EQ,D
```

then the SF command procedure is bid to display the file represented by the synonym \$EX\$IP. The .EXIT primitive then terminates the execution of the EX command procedure. If the .IF comparison is false, execution of the command procedure continues with the primitives and commands following the .ENDIF primitive for that .IF comparison.

3.5.6 .EVAL Primitive

The .EVAL primitive evaluates a numeric expression, converts the result to decimal or hexadecimal ASCII format, and stores it as the value of a specified synonym. The .EVAL primitive has the following format:

```
.EVAL [mode][= YES/NO],name = value
```

The *mode* parameter must be the keyword DEC (decimal) or HEX (hexadecimal), to specify the conversion mode. If you specify the *mode* parameter with one of these keywords and enter Y in response, the mode specified is the numeric base to which the result is converted. If you enter N, the result is converted into the mode that you did *not* specify. You can also enter the *mode* parameter without a Y/N response; in this case, the Y is assumed. The *mode* parameter is not required and automatically defaults to the decimal mode if it is not specified.

The *name* parameter specifies the name of the synonym to which the resulting value is assigned.

The *value* parameter is the numeric decimal or hexadecimal integer expression to be evaluated. Synonyms can be assigned numeric values and used as operands in the arithmetic expression. The valid arithmetic operators are:

+	unary plus or addition
-	unary minus or subtraction
*	multiplication
/	division

The character K can also be used within an expression to denote the constant value 1024.

The following examples illustrate valid .EVAL primitives. (Assume the synonym THREE has a value of 3 and the synonym TWO has a value of 2, the value shown in parentheses to the right of the example is the value assigned to the synonym RESULT.)

```
.EVAL DEC = Y,RESULT = @THREE*5 - @TWO      (13)
.EVAL DEC = N,RESULT = @THREE*5 - @TWO      (> D)
.EVAL DEC,RESULT = @THREE*5 - @TWO          (13)
.EVAL RESULT = @THREE*5 - @TWO              (13)
.EVAL HEX = Y,RESULT = @THREE*5 - TWO        (> D)
.EVAL HEX = N,RESULT = @THREE*5 - TWO        (13)
.EVAL HEX,RESULT = @THREE*5 - TWO            (> D)
```

The .EVAL primitive is useful in establishing counters for loop primitives. Refer to the paragraph concerning the .LOOP, .UNTIL, .WHILE, and .REPEAT primitives for an example of its use.

3.5.7 .LOOP, .UNTIL, .WHILE, and .REPEAT Primitives

Use the .LOOP, .UNTIL, .WHILE, and .REPEAT primitives to repeat groups of SCI statements within command procedures to form a loop. Use the .LOOP primitive to begin a loop and the .REPEAT primitive to terminate it. You can use the .UNTIL or .WHILE primitives at any point between the .LOOP and .REPEAT primitives. The loop primitives have the following format:

```
.LOOP
.UNTIL      op1, relation, op2
.WHILE      op1, relation, op2
.REPEAT
```

The .UNTIL and .WHILE primitives each contain two operands and a *relation* parameter which is a numeric or string compare operation of the type described with the .IF primitive. *Op1* and *op2* parameters can be strings, variables, or concatenated strings.

The basic structure of a loop in an SCI command procedure is:

```
.LOOP
      .
      SCI statements
      .
      .UNTIL      or .WHILE
      .
      SCI statements
      .
      .REPEAT
      .
      .
      .
```

The `.LOOP` primitive begins the loop and the `.REPEAT` primitive ends it. The loop must contain at least one `.WHILE` or `.UNTIL` primitive at any point within the loop. The SCI statements within the loop are executed until the condition of the `.WHILE` primitive is false or until the condition of the `.UNTIL` primitive is true. When either of these conditions is met, SCI executes the first statement following the `.REPEAT` primitive.

If multiple `.UNTIL` and `.WHILE` primitives are contained within a loop, SCI discontinues the loop when the first `.UNTIL` or `.WHILE` condition becomes true or false, respectively. SCI then continues with execution following the `.REPEAT` primitive.

You can also use the `.IF` primitive within the loop primitives. However, the total depth of nested loops with nested `.IF` statements cannot exceed 32.

The following example contains the loop primitives used within a command procedure.

```
.PROC  EX(EXAMPLE PROC)=0,
      INPUT PATHNAME=ACNM("@$EX$IP"),
      OUTPUT PATHNAME=ACNM("@$EX$OP"),
      DISPLAY OR COPY?=ELEMENT(D=D,C=C)(DISPLAY),
      PRINT THE FILE?=ELEMENT(Y=Y,N=N)(NO),
      LISTING DEVICE=NAME("@$EX$L"),
      NUMBER OF COPIES?=INT("@$NUM")
.SYN  $EX$IP("&INPUT PATHNAME"
.SYN  $EX$OP("&OUTPUT PATHNAME"
.IF   &DISPLAY,EQ,D
      SF FILE=@$EX$IP
      .EXIT
.ELSE
      CC IAN=@$EX$IP,OAN=@$EX$OP
.IF   &PRINT,EQ,Y
.SYN  $EX$L=&LIST
.SYN  $NUM("&NUMBER OF COPIES"
.LOOP
.UNTIL @$NUM,EQ,0
PF    FILE=@$EX$IP,L=@$EX$L
```

```
.EVAL $NUM=@$NUM-1
.REPEAT
.SYN $NUM=""
.PROMPT (SUPPLEMENTARY QUESTION),
        DELETE FILE?=ELEMENT(Y=Y,N=N)(NO)
.IF    &DF,EQ,Y
        DF PATHNAME=@$EX$IP
.ENDIF
.ENDIF
.ENDIF
.EOP
```

In the preceding example, the combination of the .LOOP, .UNTIL, .EVAL, and .REPEAT primitives creates a counter mechanism.

If the .IF comparison of this command procedure is true:

```
.IF &PRINT, EQ, Y
```

the .SYN primitives are performed, the synonym \$EX\$L is given the value of the LISTING DEVICE response, and the synonym \$NUM is given the value of the NUMBER OF COPIES? response. The loop is initiated and the .UNTIL primitive compares the value of \$NUM with 0. If \$NUM is nonzero, the Print File (PF) command procedure is bid and prints the file specified by INPUT PATHNAME to the LISTING DEVICE specified. The .EVAL primitive then decrements the value of \$NUM by one and the .REPEAT primitive causes the loop to repeat. When the value of \$NUM is 0, SCI execution continues with the statements immediately following the .REPEAT primitive.

3.5.8 .SPLIT Primitive

The .SPLIT primitive splits a value list in two, assigning the first part to one synonym and the second part to another. The primitive can have either of the following formats:

```
.SPLIT LIST = (list)[,FIRST = name][,REST = name]
or
LIST = "string"[,FIRST = name][,REST = name]
[,CHARACTER = "string" ][,POSITION = int][,STATUS = name]
```

Use the first format to split items separated by commas. Use the second format when the items are separated by characters other than commas.

3.5.8.1 Using the First .SPLIT Format. In the first format, the LIST parameter contains one value or a list of values. The first item of the list is assigned as the value of the synonym name of the FIRST parameter and the remainder of the list is assigned as the value of the synonym name of the REST parameter.

The operation of the .SPLIT primitive is as follows:

LIST	FIRST	REST
(A,B,C)	A	(B,C)
A	A	null
null	null	null
((X,Y),Z,G)	(X,Y)	(Z,G)

Items in the value list must be separated by commas. Parentheses can be used to control the splitting of the list.

The following example illustrates the first format as used in a command procedure:

```
.PROC  EX(EXAMPLE PROC)=0,
      INPUT PATHNAME=ACNM("@$EX$IP"),
      OUTPUT PATHNAME(S)=(ACNM)
.SYN   $EX$IP("&INPUT PATHNAME"
.SYN   $EX$OP="(&OUTPUT PATHNAME)"
.LOOP
.WHILE "$EX$OP",NE,$EX$OP
.SPLIT LIST="$EX$OP",
      FIRST=EXOUT,
      REST=$EX$OP
CC     IAN="$EX$IP",OAN="@EXOUT"
.REPEAT
.SYN   EXOUT=""
.EOP
```

In this example, the file specified with the INPUT PATHNAME prompt is copied to the file(s) specified with the OUTPUT PATHNAME(S) prompt.

The .SPLIT primitive within the loop determines the current output file to which the input file is to be copied. When one copy is complete, .SPLIT updates the current output file to the next output file specified in response to the OUTPUT PATHNAME(S) prompt.

The .WHILE primitive within the loop ensures that the input file is copied to all of the specified output files. When all copy processes are complete, execution continues with the primitives and commands following the .REPEAT primitive.

3.5.8.2 Using the Second .SPLIT Format. In the second format, the LIST parameter contains a character string.

The CHARACTER keyword contains a single character or a list of characters. The first occurrence of any of the characters in the list causes a split to occur. What precedes the character and the character itself go to FIRST; what remains goes to REST. Specifying by CHARACTER is useful for splitting pathnames whose nodes are separated by periods or colons.

The POSITION keyword is an integer value that specifies a character within the string. The integer value determines after which character in the string the split will occur. A 0 or negative value causes the FIRST synonym to be null and the REST synonym to contain the entire string. A value equal to or greater than the character length of the string causes FIRST to contain the entire string and REST to be null. Specifying POSITION is useful for splitting files that follow a naming convention, where you need to break off a certain number of leading characters and where any character could be found at the location of the split.

When writing the .SPLIT primitive you may indicate CHARACTER, POSITION, or both. When CHARACTER and POSITION are both present, the first condition to be satisfied determines the location of the split.

The STATUS keyword is a synonym that will be set in one of the following ways:

- For a split by CHARACTER, the synonym is set to the character on which the split occurred (namely, the character specified).
- For a split by POSITION, the synonym is set to the character preceding the split.
- For a split that does not occur, the synonym is set to null.

The following example illustrates the second format as used in a command.procedure. The synonym NAME already has the value DB1.PAYMENT.

```
.SPLIT LIST="@NAME",FIRST=DIR, REST=NAME,
      CHARACTER=(".",":"), STATUS=CHR
```

This command line assigns the value DB1 to the synonym DIR and assigns the value PAYMENT to the synonym NAME. CHR, the synonym for the STATUS keyword, is set to ".".

In the following example, the synonym NAME already has the value ABCDEFG.

```
.SPLIT LIST="@NAME",FIRST=PART1, REST=PART2,
      POSITION=3, STATUS=CHR
```

This command line assigns the value ABC to the synonym PART1 and assigns the value DEFG to the synonym PART2. CHR, the synonym for the STATUS keyword, is set to "C".

In the next example, the command line specifies a split by both CHARACTER and POSITION.

```
.SPLIT LIST="@NAME",FIRST=SITE, CHARACTER=":",
      POSITION=8, STATUS=CHR
```

This command line breaks the contents of NAME at the first colon or after the eighth character, whichever condition is satisfied first. The synonym SITE receives the first part of NAME. Since REST is not specified, the second part of NAME is discarded. CHR will be set to ":" or the eighth character of NAME.

3.5.9 .BID Primitive

The .BID primitive specifies the execution of a DNOS task. Tasks initiated with the .BID primitive share synonyms and logical names with SCI and must execute serially (that is, such tasks can only execute at a terminal one at a time).

The .BID primitive has the following format:

```
.BID TASK = name/int [,LUNO = int] [,CODE = int]
  [,PROGRAM FILE = acnm] [,PARMS = (string...string)] [,UTILITY]
```


The `TASK = name/int` parameter is required and identifies a task located in a program file. The value can be specified as a name or as an integer. The `LUNO = int` parameter is optional and specifies a LUNO assigned to the program file. The default value for LUNO is the LUNO assigned to the `.$SSHARED` program file. `CODE` is an optional value containing an integer from 0 through 255 that can be accessed by the task as a binary value. The default value for `CODE` is 0. `PROGRAM FILE` is the access name of the program file. If you specify `PROGRAM FILE`, you cannot specify `LUNO`. `PARMS` is optional and contains a list of character strings, separated by commas, that can be accessed by the task. `UTILITY` specifies that the program which is to be bid exists on the `.$$UTIL` utilities program file. If you specify `UTILITY`, you cannot specify either `LUNO` or `PROGRAM FILE`. SCI transfers control to the task upon encountering the `.BID` primitive. When the task terminates, SCI processes the next statement in the procedure.

The following example illustrates the `.BID` primitive:

```
.PROC    EX(EXAMPLE PROC)=0,
        INPUT  PATHNAME=ACNM("@$EX$IP"),
        OUTPUT PATHNAME(S)=(ACNM),
        PRINT  THE FILE?=ELEMENT(Y=Y,N=N)(NO)
.SYN     $EX$IP="&INPUT PATHNAME"
.SYN     $EX$OP="&OUTPUT PATHNAME"
.SYN     $EX$P="&PRINT"
        .BID  TASK=>40,PARMS=("@$EX$IP","@$EX$OP","@$EX$P")
.EOP
```

In this example, the `.BID` primitive bids a task with a task ID of > 40 residing in the `.$SSHARED` program file.

The values for synonyms `EXIP`, `EXOP`, and `EXP` are set to be the responses entered for the `INPUT PATHNAME`, `OUTPUT PATHNAME(S)`, and `PRINT THE FILE?` prompts, respectively. The task uses these synonym values as parameters during execution.

3.5.10 .DBID Primitive

The `.DBID` primitive specifies the execution of a task as a background task in a suspended state. The `.DBID` primitive enables you to debug the command processor using the SCI debugger. A snapshot of synonyms and logical names is taken when a task is executed by `.DBID`, allowing you to execute another foreground task concurrently.

The `.DBID` primitive has the following format:

```
.DBID TASK = name/int [, LUNO = int] [, CODE = int]
[, PROGRAM FILE = acnm] [, PARMS = (string...string)] [, UTILITY]
```

The parameter definitions of the .DBID primitive are identical to the .BID primitive parameters. The following example illustrates the .DBID primitive as it is used in a command procedure.

```
.PROC  EX(EXAMPLE PROC)=0,
      INPUT PATHNAME=ACNM("@$EX$IP"),
      OUTPUT PATHNAME(S)=(ACNM),
      PRINT THE FILE?=ELEMENT(Y=Y,N=N)(NO)
.SYN  $EX$IP("&INPUT PATHNAME"
.SYN  $EX$OP("&OUTPUT PATHNAME"
.SYN  $EX$P("&PRINT"
      .DBID TASK=USERTASK,PARMS=("@$EX$IP","@EX$OP","@$EX$P"),
      PROGRAM FILE=.USERPROG
.EOP
```

The .DBID example bids the task by name, USERTASK, from the program file .USERPROG and executes as a background task in suspended state, performing the same functions as the .BID primitive example.

3.5.11 .QBID Primitive

The .QBID primitive specifies the execution of a task as the background task of a terminal. In the interactive mode, SCI processes the next input command after initiating execution of the task and the task executes concurrently. In the batch mode, SCI is suspended until the task terminates. A snapshot of synonyms and logical names is taken when a task is executed by .QBID.

The .QBID primitive has the following format:

```
.QBID TASK = namelint [,LUNO = int] [,CODE = int]
[,PROGRAM FILE = acnm] [,PARMS = (string...string)] [,UTILITY]
```

The .QBID and .DBID primitive parameter definitions are identical. The following example uses the .QBID in a command procedure:

```
.PROC  EX(EXAMPLE PROC)=0,
      INPUT PATHNAME=ACNM("@$EX$IP"),
      OUTPUT PATHNAME(S)=(ACNM),
      PRINT THE FILE?=ELEMENT(Y=Y,N=N)(NO)
.SYN  $EX$IP("&INPUT PATHNAME"
.SYN  $EX$OP("&OUTPUT PATHNAME"
.SYN  $EX$P("&PRINT"
      .QBID TASK=USERTASK,PARMS=("@$EX$IP","@EX$OP","@$EX$P"),
      PROGRAM FILE=.USERPROG
.EOP
```

This example bids the task (USERTASK) from the program file .USERPROG and executes as a background task at the terminal, performing the same functions as the .BID primitive example.

3.5.12 .RBID Primitive

The .RBID primitive bids several system utilities. The primitive must be used with great care, so that system utilities are not affected. Refer to the *DNOS SCI and Utilities Design Document* for further information concerning the .RBID primitive.

3.5.13 .DATA and .EOD Primitives

The .DATA primitive copies data directly to a file from the command input stream. You must use the .EOD primitive to terminate the data stream. The .DATA primitive has the following format:

```
.DATA [acnm][,EXTEND[ = YES/NO]][,SUBSTITUTION[ = YES/NO]]
[,REPLACE[ = YES/NO]]
```

The *acnm* specifies the file or device to which the data is to be copied. If the *acnm* is not specified, the Terminal Local File (TLF) is assumed. Since the TLF is displayed when the command procedure completes execution, the .DATA primitive can be used to send status and error messages to the bidding terminal and place debugging statements in a procedure to track the path of execution.

There are three parameters (EXTEND, SUBSTITUTION, and REPLACE) which affect the copying process. The EXTEND parameter specifies whether the data file is to be opened extended. This allows you to concatenate several data streams under one pathname. If you do not specify the EXTEND parameter, the default is taken and the data file is not opened extended. If you use the TLF, the file is always opened extended.

The SUBSTITUTION parameter specifies whether textual substitution is to be done on the data stream before it is copied to the specified *acnm*. Textual substitution causes the appropriate values to be substituted for field prompts preceded by ampersands (&) and synonyms preceded by at signs (@). Multiple blanks are also compressed to a single blank unless they are enclosed by quotation marks (""). Any characters after an exclamation mark (!) and any comment lines are omitted. If you do not specify SUBSTITUTION, the default is taken and textual substitution is not performed.

You can use the REPLACE parameter in a data stream to replace an existing file specified by *acnm*. If you do not specify REPLACE, the default is taken and file replacement is performed. If you use the TLF, the REPLACE parameter is ignored. If you specify both EXTEND and REPLACE, EXTEND is used if REPLACE = NO.

Following is an example of the .DATA and .EOD primitives:

```
.PROC  EX(EXAMPLE PROC)=0,
        INPUT  PATHNAME=ACNM("@$EX$IP"),
        OUTPUT PATHNAME=(ACNM)
.SYN   $EX$IP("&INPUT PATHNAME"
.SYN   $EX$OP("&OUTPUT PATHNAME)"
.LOOP
.WHILE "$EX$OP",NE,$EX$OP
.SPLIT LIST="$EX$OP",
        FIRST=EXOUT,
        REST=$EX$OP
        CC IAN="@@$EX$IP",OAN="@@EXOUT"
.DATA
COPY COMPLETED
.EOD
.REPEAT
.EOP
```

In this example, the .DATA and .EOD primitives are used to output the following message to the TLF when an input file has been copied to a specified output file:

```
COPY COMPLETED
```

In the previous example, if the message was to be appended to the contents of a file (for instance, .KCOUT), the following EXTEND option must be specified for the .DATA primitive:

```
.DATA .KCOUT,EXTEND=YES
COPY COMPLETED
.EOD
```

3.5.14 .STOP Primitive

The .STOP primitive terminates execution of SCI and has the following format:

```
.STOP [TEXT = string],[CODE = int]
```

In batch mode, the *string* specified by the TEXT parameter is optional and can be used to create a message to send to the interactive SCI in place of the message BATCH SCI HAS COMPLETED. The CODE parameter is optional and is used to set the synonym \$\$BC in the synonym table of the bidding task (SCI) when a batch stream completes. The \$\$BC synonym is deleted by SCI when a background task is initiated. The TEXT and CODE parameters are ignored when SCI is not in batch mode.

You cannot enter the .STOP primitive interactively at a terminal if any of the following operations are in progress at the terminal:

- Text editing
- Debugging
- Execute System Configuration Utility (XSCU) session
- Background activity

When the .STOP primitive is processed, SCI execution does not terminate immediately. The M\$01 procedure executes, saving synonyms and logical names (in the same way the Q command does). Refer to the *System Command Interpreter (SCI) Reference Manual* for details about the M\$01 procedure.

The following example illustrates the interactive use of the .STOP primitive:

```
.PROC LO(TERMINAL IS LOGGING OFF)=0 !FULL NAME DISPLAYED
.SVC D=(0200,20) !TIME DELAY SVC
.STOP !TERMINATE SCI
.EOP
```

In this example, the .STOP primitive is used within a command procedure to stop SCI and log off the terminal after a short time delay. (The .SVC primitive is explained later in this section.)

Following is an example of a command procedure written to stop a batch stream:

```
SBATCH (STOP BATCH EXECUTION),
TEXT=*STRING, CODE=*INT
.IF @$$MO, NE, 0
.EXIT
.ENDIF
SDT
.STOP TEXT("&TEXT", CODE("&CODE"
```

Note that during the processing of .STOP, the value of \$\$CC is changed. Therefore, if you use \$\$CC in the CODE = portion of .STOP, the value will be from .STOP processing, not from previous activity.

3.5.15 .USE Primitive

The .USE primitive specifies alternate procedure libraries to be used by SCI. The .USE primitive has the following format:

```
.USE [pathname 1[,pathname 2[,pathname 3[,pathname 4 [,pathname 5]]]]]
```

From one to five pathnames follow the .USE statement. Each pathname specifies a command library. Once invoked, the menus and command procedures are taken from these libraries. If a menu or command procedure is not found after searching the library specified by *pathname 1*, then the library specified by *pathname 2* is searched, and so on. The .USE primitive remains in effect until it is replaced by another .USE or until a log-off/log-on sequence occurs.

To revert to the standard system library, specify a .USE primitive with no operands. In this case, the default value .\$\$CMDS is taken as *pathname 1* and the remaining pathnames are null.

The synonym \$\$CL is set each time a .USE is issued. The value assigned to \$\$CL is the string of libraries specified in the .USE statement. When installing a command procedure, SCI places the command definition into the directory specified by *pathname 1*. One of the pathnames must contain the main menu specified by the .OPTION primitive discussed in subsequent paragraphs.

NOTE

If the main menu cannot be found after executing the .USE primitive, a warning message is output. You must execute another .USE primitive to specify the correct command procedure directory containing the main menu.

The .USE primitive affects only the SCI session in which it was executed. Every SCI session begins with the system library .S\$CMDS as the default library. A .USE primitive executed from an interactive terminal has no effect on any batch SCI executed from the terminal.

The following example illustrates the .USE primitive for the interactive mode:

```
[ ] .USE .USERLIB, .S$CMDS
```

This statement enables you to access the command procedures in the .USERLIB library and the .S\$CMDS system library.

In the following example, the .USE primitive is used in a command procedure to access commands in the directory .USERLIB. When the .PROC primitive is encountered, SCI installs the EX (Example Proc) command procedure into the .USERLIB library.

```
.USE      .USERLIB, .S$CMDS
.PROC     EX(EXAMPLE PROC)=0,
          INPUT PATHNAME=ACNM("&$EX$IP"),
          OUTPUT PATHNAME(S)=(ACNM),
          PRINT THE FILE?=ELEMENT(Y=Y,N=N)(NO)
.SYN     $EX$IP("&INPUT PATHNAME"
.SYN     $EX$OP("&OUTPUT PATHNAME)"
.SYN     $EX$P("&PRINT"
.BID     TASK=USERTASK, PARMS=("&$EX$IP", "&$EX$OP", "&$EX$P"),
          PROGRAM FILE=.USERPROG
.EOP
```

3.5.16 .OPTION Primitive

The .OPTION primitive enables you to modify some basic interface characteristics of SCI to suit your local language or application requirements.

The .OPTION primitive has the following format:

```
.OPTION [PROMPT[ = string]][, MENU[ = name]][, PRIMITIVES[ = YES/NO]]
        [, LOWERCASE[ = YES/NO]]
```

The parameter definitions are as follows:

Keyword	Assigned Value	Function
PROMPT	An alternative prompt character string which must be less than 50 characters in length.	Enables you to specify the SCI prompt. The default SCI prompt is [] represented by the ASCII codes >7B and >7D.
MENU	Main menu name. (SCI will automatically prefix the specified menu with M\$ to obtain the menu file name in your PROC library).	Enables you to specify your own main menu which is displayed after each menu display cycle in VDT mode. The default SCI menu is named LC.
PRIMITIVES	YES or NO	Enables you to allow or disallow the use of primitives at the primary level. In the case of interactive SCI, you are allowed or disallowed to use primitives at the keyboard. In batch SCI, you are either allowed or disallowed the use of primitives in the batch stream. The default is YES.
LOWERCASE	YES or NO	Enables you to allow or disallow lowercase to uppercase mapping of input to SCI. The default is NO. Use of the LOWERCASE option does not apply to batch stream processing or the following SCI command processors: <div style="text-align: right; margin-right: 100px;"> CVD DCOPY INV MS XANAL MVI </div>

The following example shows the use of the .OPTION primitive to display the station number. MYMENU is a user constructed menu which is to be displayed, replacing the SCI main menu.

```
.PROC NP(NEW PROMPT PROC)
.OPTION PROMPT="ST@$$ST",MENU=MYMENU,PRIMITIVES=YES
.EOP
```

The synonym \$\$ST in this example represents the station number of the user's terminal.

In the following example, the .OPTION primitive is used to select the EDIT menu found in .S\$CMDS.M\$EDIT for display at each menu display cycle of SCI. This command also disables primitives at the primary level.

```
.OPTION MENU=EDIT,PRIMITIVES=NO
```

The next example illustrates the use of the .OPTION primitive to select a user-constructed menu (PROPRE) found in the user command library with the file name .M\$PROPRE, replacing the SCI main menu displayed. This command also enables the use of lowercase characters as inputs to SCI.

```
.OPTION MENU=PROPRE,LOWERCASE=YES
```

3.5.17 .MENU Primitive

The .MENU primitive causes SCI to display a specified menu the next time SCI is in command mode. The .MENU primitive has the following format:

```
.MENU [menu name]
```

There are three variations of the *menu name* parameter:

- No menu specified — The use of a .MENU with no menu specified causes screen contents to remain unchanged in the next menu cycle.
- *Menu name* — If you specify a *menu name* (one through six alphanumeric characters), SCI will display the menu in the next menu cycle, whether the station is in TTY or VDT mode. SCI appends the characters M\$ at the beginning of the name to obtain the file name within your command procedure library file where the menu resides.
- **Menu name* — If you specify a *menu name* preceded by an asterisk, the menu is displayed only if the station is in VDT mode.

An equivalent alternative to the .MENU primitive is the slash (/) symbol. It is defined as follows:

```
/ is equivalent to .MENU
/DEV is equivalent to .MENU DEV
/*DEV is equivalent to .MENU *DEV
```


The following example illustrates the .MENU primitive:

```
.PROC    NM(NEW MENU PROC)
.MENU    MYMENU
.EOP
```

This example specifies MYMENU to be displayed on the terminal. SCI searches the command library for the file M\$MYMENU and displays the menu in that file on the next menu display cycle.

3.5.18 .SHOW Primitive

The .SHOW primitive displays the contents of a specified file or files to an interactive terminal or batch listing file. The .SHOW primitive has the following format:

```
.SHOW filename[,filename...filename]
```

where:

filename is the name of a file.

The .SHOW primitive cannot display program files, image files, or directories.

.SHOW is equivalent to the Show File (SF) command. The function keys used with the SF command are applicable to the .SHOW primitive. Refer to the SF command in the *System Command Interpreter (SCI) Reference Manual* for descriptions of the function keys.

The following example illustrates the .SHOW primitive used in the SF command procedure:

```
.PROC    SF(SHOW FILE)=0,
          INPUT FILENAME=ACNM("@$$F$IP")
.SYN     $$F$IP("&INPUT FILENAME"
.SHOW    @@$$F$IP
.EOP
```

In the previous example, the .SHOW primitive displays the file specified as the response to the INPUT FILENAME prompt. The .SHOW primitive is not limited to the SF command procedure; it can be used in any command procedure.

3.5.19 .SVC Primitive

The .SVC primitive allows you to issue supervisor calls (SVCs) from the SCI procedure language. The format of the .SVC primitive is as follows:

```
.SVC[$name] DATA/BYTE/TEXT = value(s)...[$name] DATA/ BYTE/TEXT = value(s)
```

The optional *\$name* parameter is a synonym that can be used to retrieve information returned to the SVC call block by DNOS. The DATA, BYTE, and TEXT parameters enable you to describe the SVC call block with *value(s)* as they might appear in assembly language. The execution of this command causes SCI to build the supervisor call block and issue the SVC. The synonym \$\$CC is set to >00 if the SVC completes normally; otherwise, SCI sets an error condition enabling the command procedure to test for abnormal cases via the \$\$CC and \$\$MN synonyms. The \$\$CC and \$\$MN synonyms are described elsewhere in this manual.

There are some SVCs which cannot be used with the .SVC primitive. These limitations are significant and are described in Table 3-8.

To use the .SVC primitive, perform the following steps:

1. Determine the SVC to be issued. Make sure that it does not fall into any of the categories of disallowed SVCs.
2. Format the SVC call block using the DATA, BYTE, and TEXT parameters. After each parameter, insert an equals sign (=).
 - a. If the SVC definition requires a pointer to a text string, replace the pointer (which is always a DATA value) with the text. The following example illustrates this use in the System Log SVC when written in assembly language:

```

SVCB      DATA >2100
          DATA 0
          DATA POINTER
          DATA 0
POINTER  BYTE 20
          TEXT 'TEXT FOR LOG MESSAGE'
```

The following is an example of .SVC primitive used to issue the System Log SVC:

```

.SVC      DATA=>2100,
          DATA=0,
          DATA="TEXT FOR LOG MESSAGE",
          DATA=0
```

SCI realizes the text supplied for the pointer is not a data value and stores the text string with the preceding byte length count. A pointer to this string is generated by SCI and placed in the SVC block.

- b. If the SVC definition requires text within the call block, you must declare the field length of the text. To do this, place the field length within parentheses before the text. Following is an example of this use in the Map Task Name to ID SVC when written in assembly language:

```

SVCB      DATA >3100
          DATA 0
          TEXT  SCI990
          DATA 0FF00
          DATA 0
```

The following is an example of the .SVC primitive used to issue the Map Task Name to ID SVC:

```
SVC          DATA=>3100,
             DATA=0,
             TEXT=(8)SCI990,
             DATA=0FF00,
             DATA=0
```

This example causes the field to be blank filled with the text left-justified.

3. If you are using a list of DATA or BYTE values, enclose the list in parentheses, as shown below for the System Log Message SVC:

```
.SVC        DATA=(>2100,0,"MESSAGE FOR SYSTEM LOG",0)
```

A list of TEXT is not allowed.

4. Information returned by DNOS is retrieved from the call block by placing a label name before the DATA, BYTE, or TEXT field of interest. The label name must begin with a \$, as shown below when issuing the Map Task Name to ID SVC using the .SVC primitive:

```
.SVC        DATA=(>3100,0),
             TEXT=(8)SCI990,
             BYTE=0FF,
$TASKID     BYTE=0,
             DATA=0
```

After the SVC is issued, SCI assigns the hexadecimal ASCII value of the DATA or BYTE parameter or the text string of the TEXT parameter to the system \$TASKID. If a list of values was specified as the DATA or BYTE parameter, the synonym is assigned the first item of the list. Such synonyms are assigned whether or not an error occurs when the SVC is executed.

The following examples illustrate uses of the .SVC primitive.

The first example uses the .SVC primitive to cause a time delay of 100 system time units, as shown below:

```
.SVC      DATA=(0200,100)
```

The second example uses the .SVC primitive to kill a task with a run ID value equivalent to the value of the synonym \$\$RI. The state of the task terminated is converted to ASCII (base 16) and the synonym \$STATE is assigned that value.

```
.SVC      DATA=03300,
          BYTE=(@$RI,0),
          $STATE BYTE=(0,0,0,0)
```

In the third example, the .SVC primitive is used to assign a global LUNO to the directory .S\$CMDS and return the LUNO that was assigned as the value of \$\$LU. A special feature of the DATA field is shown here. If any element in the DATA list cannot be converted into a number, SCI allocates memory and copies the element as a string (preceded by a byte count), placing the address of the string in the SVC block.

```
.SYN TYPE=02000          ! TYPE IS DIRECTORY FILE
.SYN SCOPE=01000        ! THE LUNO SCOPE IS GLOBAL
.SVC      DATA=0,
          BYTE=>91,
$$LU     BYTE=0,
          DATA=(0,0,0,0,0,0,@TYPE+@SCOPE+0400,0,0),
          DATA=.S$CMDS
```

The advantages of the .SVC primitive as opposed to bidding a separate task to perform the SVC are as follows:

- Efficiency — The overhead involved to bid a task is avoided.
- Generality — Some functions are supported by the operating system and are not available in the standard set of SCI commands.

Table 3-8. Disallowed SVCs for .SVC Primitive

SVC Type	Restrictions		
Privileged SVCs	<p>Since SCI is not a privileged task, privileged SVCs cannot be issued. DNOS enforces this limitation. Refer to the <i>DNOS Supervisor Call (SVC) Reference Manual</i> to determine if the desired SVC is privileged.</p>		
External Data Blocks	<p>SVCs that use external data blocks (other than text string format for input) are not allowed. SCI enforces this limitation and disallows the following SVCs:</p>		
	SVC	Subopcode	Function
	> 00	> 05	Read Characteristics
		> 09	Read ASCII
		> 0A	Read Direct
		> 0B	Write ASCII
		> 0C	Write Direct
		> 10	Rewrite
		> 40 – > 52	All KIF Subopcodes
	> 03		Get Date and Time
	> 0A		Convert Binary to
			Decimal
	> 0B		Convert Decimal to
			Binary
	> 0C		Convert Binary to Hexa-
			decimal
	> 0D		Convert Hexadecimal to
			Binary
	> 1C		Put Data
	> 1D		Get Data
	> 3B		Initialize Date and Time
	> 3F		Retrieve System Data
	> 45		Encrypt Data
	> 46		Decrypt Data
	> 47		Log Accounting Entry
	> 4C		Return Code Processor

Table 3-8. Disallowed SVCs for .SVC Primitive (Continued)

SVC Type	Restrictions																						
Special SVCs	SVCs that might jeopardize the internal functioning of SCI are not allowed. SCI enforces this limitation and disallows the following SVCs:																						
	<table border="0"> <thead> <tr> <th data-bbox="610 558 666 585">SVC</th> <th data-bbox="928 558 1029 585">Function</th> </tr> </thead> <tbody> <tr> <td data-bbox="610 617 662 644">> 01</td> <td data-bbox="928 617 1055 644">Wait for I/O</td> </tr> <tr> <td data-bbox="610 646 662 674">> 04</td> <td data-bbox="928 646 1105 674">Terminate Task</td> </tr> <tr> <td data-bbox="610 676 666 703">> 0F</td> <td data-bbox="928 676 1140 703">Abort I/O by LUNO</td> </tr> <tr> <td data-bbox="610 705 662 732">> 10</td> <td data-bbox="928 705 1079 732">Get Common</td> </tr> <tr> <td data-bbox="610 735 662 762">> 12</td> <td data-bbox="928 735 1070 762">Get Memory</td> </tr> <tr> <td data-bbox="610 764 662 791">> 13</td> <td data-bbox="928 764 1120 791">Release Memory</td> </tr> <tr> <td data-bbox="610 793 662 821">> 14</td> <td data-bbox="928 793 1079 821">Load Overlay</td> </tr> <tr> <td data-bbox="610 823 666 850">> 1B</td> <td data-bbox="928 823 1130 850">Release Common</td> </tr> <tr> <td data-bbox="610 852 662 879">> 40</td> <td data-bbox="928 852 1140 879">Segment Manager</td> </tr> <tr> <td data-bbox="610 882 662 909">> 43</td> <td data-bbox="928 882 1105 909">Name Manager</td> </tr> </tbody> </table>	SVC	Function	> 01	Wait for I/O	> 04	Terminate Task	> 0F	Abort I/O by LUNO	> 10	Get Common	> 12	Get Memory	> 13	Release Memory	> 14	Load Overlay	> 1B	Release Common	> 40	Segment Manager	> 43	Name Manager
SVC	Function																						
> 01	Wait for I/O																						
> 04	Terminate Task																						
> 0F	Abort I/O by LUNO																						
> 10	Get Common																						
> 12	Get Memory																						
> 13	Release Memory																						
> 14	Load Overlay																						
> 1B	Release Common																						
> 40	Segment Manager																						
> 43	Name Manager																						

3.6 SCI PRIMITIVE BATCH STREAM EXAMPLE

The following example uses SCI primitives in a batch stream to install a program in a program file. The first command of every batch stream should be the Batch SCI (BATCH) command and the last command should be the End Batch SCI (EBATCH) command. The BATCH command clears unnecessary synonyms and EBATCH indicates that there are no more commands to be processed in the batch stream.

EXAMPLE

```

BATCH LS=YES
*****
*
* ASSEMBLE, LINK AND INSTALL THE EXAMPLE TEST TASK
* ASSUME: SOURCE = SOURCE DIRECTORY
*         OBJECT = OBJECT DIRECTORY
*         LISTING = LISTING DIRECTORY
*         PROG   = PROGRAM FILE
*         CONTROL = LINK CONTROL DIRECTORY
*         LINK   = LINKED OUTPUT DIRECTORY
*         LINKMAP = LINKMAP DIRECTORY
*
*****
*
* ASSEMBLE EXAMPLE SOURCE MODULES: EXAMPLE01, EXAMPLE02
*
XMA SOURCE=SOURCE.EXAMPLE01, OBJECT=OBJECT.EXAMPLE01,
    LISTING=LIST.EXAMPLE01, OPTIONS=(XREF,DUN,BUN,TUN)
EC
XMA SOURCE=SOURCE.EXAMPLE02, OBJECT=OBJECT.EXAMPLE02,
    LISTING=LIST.EXAMPLE02, OPTIONS=(XREF,DUN,BUN,TUN)
EC
*****
*
* IF NO ASSEMBLY ERRORS THEN LINK THE EXAMPLE TASK
*
.IF @$E$C,EQ,0
XLE CONTROL=CONTROL.EXAMPLE, LINKED OUTPUT=LINK.EXAMPLE,
    LISTING=LINKMAP.EXAMPLE
EC
.ENDIF
*****
*
* IF NO ERRORS OCCURRED DURING THE LINK
* DELETE THE CURRENT TASK
*
.IF @$E$C,EQ,0
DT PROGRAM FILE=PROG, TASK NAME=EXAMPLE
.ENDIF
*****
*
* IF NO ERRORS OCCURRED DELETING THE OLD TASK
* INSTALL THE NEW PROGRAM
*
.IF @$E$C,EQ,0
IT PROGRAM FILE=PROG, OBJECT PATHNAME=LINK.EXAMPLE
EC
.ENDIF
*****
*
* INSTALLATION OF EXAMPLE PROGRAM COMPLETE
*
EBATCH TEXT="INSTALLATION COMPLETE, ERRORS=@$E$C",CODE=@$E$C
    
```

```

*
*      IF NO ERRORS OCCURRED DELETING THE OLD TASK
*          INSTALL THE NEW PROGRAM
*
*.IF @$E$C,EQ,0
IT      PROGRAM FILE=PROG, OBJECT PATHNAME=LINK.EXAMPLE
EC
.ENDIF
*****
*
*          INSTALLATION OF EXAMPLE PROGRAM COMPLETE
*
EBATCH TEXT="INSTALLATION COMPLETE, ERRORS=@$E$C",CODE=@$E$C

```

3.7 ERROR PROCESSING FOR PRIMITIVES

If SCI encounters a syntax error within the parameters of a primitive in a command procedure, the command procedure execution aborts. If the error occurs on a primitive that is in a batch stream, the batch stream terminates at the point of error.

If the destination file cannot be accessed for the .PROC or .DATA primitives, SCI scans until it encounters an .EOP or .EOD primitive. SCI then sets the synonym \$\$CC to an error condition and continues processing. For example, if a batch stream is executed and the destination file for a .PROC or .DATA primitive is not accessible, SCI continues to execute the remaining procedures.

3.8 COMMAND PROCEDURES AND COMMAND PROCESSORS

In addition to the command procedures and command processors supplied with the DNOS operating system, the system programmer can write new procedures and processors which fulfill user requirements. The rules of the SCI language syntax and the SCI primitives discussed earlier in this section are applied when writing new procedures and processors. The following paragraphs discuss the design of command procedures and command processors and the various options available.

3.8.1 Command Procedure Design

A command procedure is a sequence of statements that is executed each time you issue an SCI command. The command procedure is composed of SCI statements including commands, primitives, and menu invocations executable by SCI.

Command procedures are designed to:

- Collect responses to field prompts
- Assign values to synonyms to be used by this or other procedures
- Call command processors and/or other command procedures

Procedures can collect responses to field prompts and perform various actions such as comparisons with the .IF primitive. For example, the Install Task (IT) command tests the response to the field prompt ATTACHED PROCEDURES? and may issue additional field prompts, depending on the result of the comparison.

A synonym can be enclosed in parentheses after the field prompt type and defined as the initial value of the field prompt in a command procedure, as in the following:

```
.PROC SF(SHOW FILE),
      FILE PATHNAME=*ACNM("@$$SF$P")
.SYN  $$SF$P("&FILE PATHNAME"
.IF   "&FILE PATHNAME",NE, ""
      .SHOW @&FILE PATHNAME
      .ENDIF
.EOP
```

In this procedure, the value of the synonym \$\$SF\$P is the initial value for FILE PATHNAME. The @ sign preceding the synonym designates the synonym value. When you first log on the system, \$\$SF\$P has no assigned value and the field prompt has no initial value displayed. After you execute the first SF, the synonym is assigned the value which is entered as the initial value for FILE PATHNAME in subsequent SF executions.

In addition to assigning synonyms within the command procedure, the use of a synonym as an initial value enables the command procedure to recall the synonym with its last assigned value. Initial values are not required to be synonyms in command procedures; they can be numbers or strings. For initial values that are synonyms, values are not required to be assigned for the synonyms. For example, if the synonym ,FILE has no value assigned, the character string FILE following the , sign is assigned as the value.

After all references are resolved, if the initial value for a field prompt begins with a \$ sign, then the field prompt is given a null value (not the string that begins with the \$ sign as the initial value).

The ability of one procedure to call other procedures offers you several advantages. It allows one procedure to perform a simple field prompt test and branch to other command procedures, thus simplifying user input. As an example of such a procedure, the CF command can be written as follows:

```
.PROC CF (CREATE FILE - SEQ, REL, KEY, DIR, PRO, IMG),
      FILE TYPE=ELEMENT(SEQ, REL, KEY, DIR, PRO, IMG) (SEQ)
      *
.IF   "@$$M0", EQ, 0
  MSG   T="ERROR: USE SPECIFIC FILE TYPE CREATE IN BATCH"
.ELSE
.IF   "&FILE TYPE", EQ, SEQ
  CFSEQ
.ENDIF
.IF   "&FILE TYPE", EQ, REL
  CFREL
.ENDIF
```

```

        .IF      "&FILE TYPE", EQ, KEY
        CFKEY
        .ENDIF
        .IF      "&FILE TYPE", EQ, DIR
        CFDIR
        .ENDIF
        .IF      "&FILE TYPE", EQ, PRO
        CFPRO
        .ENDIF
        .IF      "&FILE TYPE", EQ, IMG
        CFIMG
        .ENDIF
        .ENDIF
.EOP

```

In the batch mode of SCI operation, the top level procedure of a nested command definition must recognize all field prompts of the command, including those used by the nested procedure. In the following example, the field prompt OBJECT ACCESS NAME is used in the nested procedure and must be specified in the top level procedure.

```

.PROC  RUN(EXECUTE APPLICATION PROGRAM),
        LANGUAGE(COBOL,PASCAL)=NAME,
!      XCP PROMPT
        OBJECT ACCESS NAME
!      CHECK FOR BATCH MODE
        .IF "@$$MO",EQ,0
        .IF "&LANGUAGE",EQ,"COBOL"
XCP  OBJECT ACCESS NAME="&OBJECT ACCESS NAME"
        .ENDIF
        .EXIT
        .ENDIF
!      INTERACTIVE MODE
        .IF "&LANGUAGE",EQ,"COBOL"
XCP
        .EXIT
        .ENDIF
        .IF "&LANGUAGE",EQ,"PASCAL"
XPT
        .EXIT
        .ENDIF
.EOP

```

The top-level procedure RUN calls the Execute COBOL Program (XCP) command; however, the RUN procedure is divided into two parts. The first part of the RUN procedure is used for batch execution. The XCP command is called with the appropriate field prompt values, assuming all synonyms are assigned correct values prior to execution.

NOTE

Synonyms can be set by previous command procedures in a batch stream. Therefore, you must take appropriate action so that the correct synonyms are used.

The second part of the RUN procedure is used for interactive mode. When the XCP command is called, field prompt values are not assigned so that they will be prompted at the terminal.

NOTE

As soon as a procedure terminates, its field prompts and their values are destroyed.

The following examples show how to call the RUN procedure in batch and interactive modes.

Example of batch mode:

```
RUN  LANG=COBOL,OBJ=O.TEMP
```

Example of interactive mode:

```
[ ]RUN
EXECUTE APPLICATION PROGRAM
      LANGUAGE: COBOL

EXECUTE COBOL PROGRAM<VERSION:3.3.1 81196>
      OBJECT ACCESS NAME: O.TEMP
      MESSAGE ACCESS NAME:
      SWITCHES: 00000000
      FUNCTION KEYS: NO
```

If a command procedure uses the .PROMPT primitive in the interactive mode to gather responses from additional screens of prompts, special provisions must be made for batch execution. The procedure must include DEFAULT type prompts for all .PROMPT screens along with the first screen of prompts. For example, the following CIC command procedure includes DEFAULT types for the RESOURCE TYPE and PROCESS ASSIGNS? prompts that are to be displayed in the second screen.

```

CIC(CREATE IPC CHANNEL)=2,
CHANNEL PATHNAME      = ACNM,
OWNER TASK PROGRAM FILE= ACNM,
OWNER TASK NAME OR ID = NAME/RANGE(0,OFF),
CHANNEL TYPE          = ELEMENT(S=SYMMETRIC,
M=MASTER/SLAVE)(SYMMETRIC),
CHANNEL SCOPE         = ELEMENT(G=GLOBAL,
T=TASK,J=JOB)(GLOBAL),
MAXIMUM MESSAGE LENGTH = RANGE(1,03000)(100),
SHARED CHANNEL ACCESS? = ELEMENT(Y=YES,N=NO)(YES),
RESOURCE TYPE         = DEFAULT(CHAN),
PROCESS ASSIGNS?      = DEFAULT(NO)
  .IF "&CHANNEL TYPE",EQ,MASTER/SLAVE
  .PROMPT(MASTER/SLAVE CHANNEL ATTRIBUTES),
RESOURCE TYPE = INT
PROCESS ASSIGNS?=ELEMENT(Y=YES,N=NO)(NO)
  .
  .
  .
  (procedure continues)

```

The field prompts from .PROMPT screens are defined without type information. This allows the use of a single command procedure in both interactive and batch modes, but prompt the field prompts in groups in interactive mode.

The following examples show how to use the CIC procedure in batch mode and interactive mode.

Example of batch mode:

```

CIC CHANNEL PATHNAME=.MKC.REPORT,
OWNER TASK PROGRAM FILE=.MKC.COMPILE,
OWNER TASK NAME OR ID=TAX,
CHANNEL TYPE=MS,
CHANNEL SCOPE=T,
MAXIMUM MESSAGE LENGTH=100,
SHARED CHANNEL ACCESS?=YES,
RESOURCE TYPE=CHAN,
PROCESS ASSIGNS?=NO

```

Example of interactive mode:

```

[]CIC
CREATE IPC CHANNEL
CHANNEL PATHNAME: .MKC.REPORT
OWNER TASK PROGRAM FILE: .MKC.COMPILE
OWNER TASK NAME OR ID: TAX
CHANNEL TYPE: MS
CHANNEL SCOPE: TASK
MAXIMUM MESSAGE LENGTH: 100
SHARED CHANNEL ACCESS: YES

```

```

MASTER/SLAVE CHANNEL ATTRIBUTES
RESOURCE TYPE: CHAN
PROCESS ASSIGNS?: NO

```

All of the field prompts and values which are defined within a procedure are stored in a table until the procedure terminates. Therefore, when a procedure calls another procedure (or itself), all of the field prompts and values for both procedures are stored in the table. As a result of deep level nesting, the table can become full and cause the command procedure to abort with the following message:

```
U SCI-0036 FIELD PROMPT TABLE OVERFLOW
```

To prevent table overflow, you should avoid deep levels of nesting and recursive procedures. To avoid nesting and still call numerous procedures, call as many procedures as possible at the same level. Procedures do not have access to one another's field prompts. Recursive procedures can usually be avoided by using iterative loops within a command procedure.

3.8.2 Command Processor Design

A command processor can be written in any language supported by DNOS. The command processor is invoked by the command procedure using the .BID or .QBID primitive to initiate the program as a foreground task or a background task, respectively. If the program is run as a foreground task, SCI execution is suspended until the task completes. Processors that involve long-term execution should operate in background, rather than foreground.

The following code is a simple example of a processor written in assembly language invoked by the command procedure EXP (defined in the next section):

EXAMPLE

```

IDT      EXPRO
*        THIS IS THE EXAMPLE COMMAND PROCESSOR CALLED BY
*        EXP.  IT GETS THE 2 PARAMETER VALUES AND RETURNS
*        THEM AS A MESSAGE IN THE MESSAGE BUFFER.  THE
*        MESSAGE IS THEN DISPLAYED WHEN THE RETURN TO THE
*        COMMAND INTERPRETER IS MADE.
REF S$GTCA,S$PARM,S$RTCA,S$STOP
DATA WS,PC,ERROR0
WS       BSS 32          PROCESSOR WORKSPACE
MSG     BYTE 255        TOTAL MESSAGE SIZE
        BSS 255        MESSAGE BUFFER
ERR0    BYTE 17
        TEXT ':END ACTION TAKEN'
ERR1    BYTE 27
        TEXT ':ERROR RETURNED FROM S$PARM'
ERR2    BYTE 27
        TEXT ':ERROR RETURNED FROM S$GTCA'
PC      BLWP @S$GTCA    GET THE TCA
        MOV R0,R0      TERMINATE
        JNE ERROR2     IF ERROR
        LI R4,MSG      GO GET

```

```

        LI    R3,1                THE FIRST
        BLWP @S$PARM             PARAMETER,WHICH IS
        BYTE R3,R4              "&EXAMPLE NAME"
        MOV   R0,R0              TERMINATE
        JNE  ERROR1             IF ERROR
        MOVB @MSG,R7            SET R7 = NUMBER OF
        SRL  R7,8               CHARACTERS READ IN
        A    R7,R4              R4 POINTS TO LAST CHARACTER
*                                OF FIRST STRING
        NEG  R7                 R7 = LENGTH OF
        AI   R7,255            REMAINING BUFFER
        MOVB *R4,R6            R6 = LAST CHARACTER OF 1ST STRING
        SWPB R7                R4 NOW POINTS TO THE
        MOVB R7,*R4            REMAINING BUFFER
        LI   R3,2              GO GET THE SECOND
        BLWP @S$PARM             PARAMETER,WHICH IS
        BYTE R3,R4              "&NUMBER"
        MOV   R0,R0              TERMINATE
        JNE  ERROR1             IF ERROR
        AB   *R4,@MSG          @MSG CONTAINS LENGTH OF MESSAGE
        MOVB R6,*R4            RESTORE LAST CHAR OF 1ST STRING
*                                NORMAL NO ERROR RETURN
        LI   R2,MSG            RETURN MESSAGE
        CLR  R1
RETURN  BLWP @S$RTCA           RELEASE TCA
        BLWP @S$STOP          RETURN TO SCI
*                                END ACTION
ERROR0  LI   R2,ERR0
        LI   R1,>8000
        JMP  RETURN
*                                ERROR RETURN FROM S$PARM
ERROR1  LI   R2,ERR1
        LI   R1,>8000
        JMP  RETURN
*                                ERROR RETURN FROM S$GTCA
ERROR2  LI   R2,ERR2
        LI   R1,>8000
        JMP  RETURN
        END

```

Note that the processor receives its parameters and returns control to SCI by calling various interface routines (S\$GTCA, S\$RTCA, S\$STOP, S\$PARM). These routines are discussed later in this section.

The command processor terminates by calling S\$STOP and control returns to the command procedure statement following the .BID primitive which invoked the processor.

3.8.3 Installing Command Procedures and Command Processors

After the command procedure and command processor (if applicable) are created, install each of them before using the new command. Install the command procedure in a procedure library. Although you can install user-defined commands in the .S\$CMDS command procedure library, it is recommended that you install them in a separate user library to prevent accidental alterations of supplied commands. Install the command processors, if used, in a program file.

You can install command procedures interactively or through batch execution. The following example creates a user command procedure library on the system disk and installs the EXP command in that library interactively:

```
[ ]CFDIR      PATHNAME=SYSVOL.USERLIB,MAXENT=101
[ ].USE       SYSVOL.USERLIB,.$$CMDS
[ ].PROC      EXP(EXAMPLE PROCEDURE),
              EXAMPLE NAME=STRING("SAMPLE"),
              NUMBER=INT
              .BID      TASK=>55,PARMS=("&EXAMPLE NAME","&NUMBER"),
              PROGRAM FILE=.USERPROG
              .EOP
[ ]
```

The Create Directory File (CFDIR) command creates the user library which is then specified in the .USE primitive. By specifying .S\$CMDS as the secondary library in the .USE primitive, the standard SCI commands are still available to use. Enter the command procedure statements following the .USE primitive. If several procedures are to be entered in the same library, additional .USE primitives are not necessary.

A batch stream is more efficient than interactive entry, particularly when defining several procedures. The sequence to create a batch stream for the above command procedure is as follows:

1. Issue the Execute Text Editor (XE) command without specifying an input FILE ACCESS NAME.
2. Edit the above SCI statements into the file. Include the BATCH and EBATCH commands as the first and last commands, respectively.
3. Issue the Quit Edit (QE) command, do not abort the edit, and specify some file (for example, MKC.NEWPROC) as the OUTPUT FILE ACCESS NAME.

Refer to the *DNOS Text Editor Reference Manual* for information about the Text Editor.

Install the command procedure by issuing the Execute Batch (XB) command, specifying MKC.NEWPROC as the INPUT ACCESS NAME. When the batch stream completes, check the listing file for errors. If no errors occurred, the command procedure is installed in the library SYSVOL.USERLIB.

The task called by the .BID primitive is the assembly code example used in the paragraphs discussing the command processor. Before installing the command processor, you must compile or assemble the processor and any routines it calls (excluding the S\$ routines, which exist in object form), then link edit the processor. If the processor uses the S\$ interface routines, the link edit control file must use the library .SCI990.S\$OBJECT containing the S\$ routines. Following is an example of the Link Editor control stream needed to link the example processor with the interface routines:

LIBRARY .SCI990.S\$OBJECT	S\$ routines
PHASE 0, MYPROC1	Name of linked object module
INCLUDE .MYPROC1	Processor object module
END	

After you link edit the processor, install it in a program file using the Install Task (IT) command.

3.8.4 Using New Commands

After you install the command procedures in the .S\$CMDS library and the command processors in program files, you can issue new commands whenever SCI prompts for a command. If you install the new commands in a user command library, you must specify that library by executing the .USE primitive before the commands are available.

The following example specifies a command library, other than .S\$CMDS, to be used by SCI:

```
[ ] .USE    SYSVOL.USERLIB
[ ] EXP
EXAMPLE PROCEDURE
      .
      .
      .
```

The last .USE primitive executed specifies the current library. Executing the .USE primitive without a pathname defaults to the system library, .S\$CMDS.

3.8.5 Expert Mode Considerations

The way SCI commands are entered interactively in expert mode is similar to the way these commands are entered in batch mode. Enter the command and answer the following prompts and responses necessary for command execution. The following is an example of the Show File (SF) command:

```
SF  F=DS02.USER.TEMP
```

After you press the RETURN or ENTER key, the file that you specified is displayed.

If a command requires additional prompt responses, separate them by commas. If you execute a command using this method and exclude a prompt necessary for command execution, SCI allows you to enter a response to that prompt. For instance, suppose SOURCE FILE, DESTINATION FILE, and LISTING FILE were the prompts for a command. Responses to the SOURCE FILE and DESTINATION FILE prompts must be supplied, but response to the LISTING FILE is optional. If you enter a response to only the SOURCE FILE prompt:

```
EX S=DS02.USER.SOURCE
```


then SCI displays the prompts of the command as if you had only entered the command name. The cursor is positioned at the prompt which needs a response, as in the following:

```

SOURCE FILE: DS02.USER.SOURCE
DESTINATION FILE: (cursor positioned here)
LISTING FILE:

```

This enables you to enter the response to the DESTINATION FILE prompt. If necessary, you can change the response to the SOURCE FILE at this point. However, SCI will not allow you to enter a response to the LISTING FILE prompt; you can only modify responses preceding the cursor position. Press the RETURN key and the command will execute.

If a response requires a list of values, this list must be enclosed in parentheses if you are using expert mode.

When you issue a command procedure interactively to call another command procedure and it does not specify any field prompts or values for the second procedure, the field prompts for the second procedure are displayed. However, if you enter a command procedure in *expert mode* to call another command procedure, the prompts are not displayed for the second procedure.

Another type of expert mode is used when the the initial or default value is to be used. Enter the command with a period as in the following:

```
SF .
```

Press the RETURN or ENTER key so the file specified as the initial or default value will be displayed. You cannot use this type of expert mode unless the nonoptional command prompts have predefined initial or default values. For example, the SF command does not have a default value and the initial value is not assigned until after the first execution of the command. Therefore, if SF has not been executed previously, the SF. command would not show a file.

3.8.6 Deleting Commands

If you want to delete a command, delete the command procedure from the command procedure library and the command processor (if applicable) from the program file in which they are installed. Before deleting the command processor, ensure that it is not called by other command procedures.

Delete a command procedure by entering a Delete File (DF) command. The following example deletes the EXP command installed in a user library:

```
DF PATHNAME=SYSVOL.MYLIB.EXP
```

Delete a command processor by using the Delete Task (DT) command.

3.9 COMMAND PROCESSOR INTERFACE ROUTINES

A processor communicates with SCI through command processor interface routines (S\$ routines). For example, a processor uses the S\$PARM routine to obtain the parameters supplied with the .BID, .DBID, or .QBID primitives. Calling S\$ routines allows you to write processors similar to those supplied with DNOS, which contain calls to S\$ routines that are transparent to the user. Since S\$ routines require an assembly language interface, high-level language users must provide assembly language routines to call them. Refer to the appropriate language manual for details on interface routines.

Several other DNOS processors provide interfaces similar to the SCI interfaces. These include an interface to the mailbox message subsystem and one to the operator interface subsystem. Routines with names beginning with MB\$ are interfaces to the mailbox message subsystem, and those beginning with OI\$ are interfaces to the system operator subsystem. The library with the S\$ routines includes these interface routines.

3.9.1 Interface Routine References

References to interface routines in a command processor are external references. Each referenced routine must be listed in a REF assembler directive in the command processor program. Branch and Link Workspace Pointer (BLWP) instructions call interface routines. Sometimes, data follows a routine, as specified in the calling sequences of the routine. The following example shows the REF directive and the BLWP instructions required in a command processor that calls the following interface routines: Get TCA (S\$GTCA), Terminate and Return to SCI (S\$TERM), and Release TCA (S\$RTCA).

```

      .
      .
REF   S$GTCA, S$RTCA, S$TERM
      .
      .
BLWP  @S$GTCA
      .
      .
BLWP  @S$RTCA
      .
      .
BLWP  @S$TERM
    
```

Any task using S\$ routines must be linked with the routines. They are located in library .SCI990, which must be specified for link editing. Tasks written for DX10 must be relinked with DNOS S\$ routines to execute on DNOS.

If the S\$ routine library for DX10 has been linked with a DNOS task, the following error message appears when the task executes:

```

***ERROR*** TASK ID >xx HAS BEEN LINKED TO DX10 S$ROUTINES
    
```

where:

xx is the installed ID of the task.

This message appears when the S\$GTCA routine or the Initialize System Data Base (S\$NEW) routine executes.

3.9.2 Buffers for Interface Routines

Many of the S\$ interface routines require buffers. The form of these buffers is as follows:

```

LABEL  BYTE  CNT
        BSS  CNT
    
```

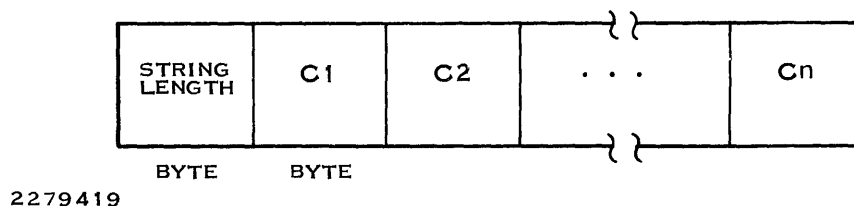
CNT represents the number of bytes in the buffer, excluding the count byte. When the buffer contains a character string, the form is as follows:

```

LABEL  BYTE  CNT
        TEXT '_____'.
CNT    EQU   $ - LABEL - 1
    
```

In this example, symbol CNT represents the number of characters in the string provided by the TEXT directive.

The resulting buffer has the following form:



where:

string length indicates the number of characters in the string.

buffer size represents one byte for each character in the string plus one byte reserved for the length count.

Unless otherwise indicated, all references to buffers in the descriptions of the interface routines refer to a buffer with the byte count in the first byte, as in the preceding examples.

If you create a buffer of length n and you designate the buffer in a routine that returns a value, the length byte will receive the length of the returned string.

3.10 INTERFACE ROUTINE DESCRIPTIONS

The six major classes of interface routines are as follows:

- SCI interface routines
- Local display file routines
- String utility routines
- Arithmetic utility routines
- Mailbox message routines
- Operator interface routines

3.10.1 SCI Interface Routines

The SCI interface routines access parameters (PARMS and CODE), locate and modify synonyms defined for the session, and return control to SCI. Only tasks bid with the .BID, .QBID, or .DBID primitives can use these routines. The communications area (TCA) serves as an information buffer between SCI and the command processors. A call to the S\$GTCA routine must precede any calls to these S\$ routines. After a call to the S\$RTCA routine, a call to any of these routines is not valid. You can call the S\$TERM routine and the Alternate Termination (S\$STOP) routine any time, since they call S\$GTCA.

3.10.1.1 Get TCA (S\$GTCA). Routine S\$GTCA makes information available to a command processor. You must call this routine before a command processor can access synonym values or get parameters passed to it by the SCI procedure. After a successful call to S\$GTCA, the processor can retrieve the values of CODE and PARMS.

Calling Sequence

```
BLWP @S$GTCA
```

Registers Used

R0 — Error code returned by S\$GTCA

Error Codes

> FF05 — Unable to access name correspondence table (NCT)

3.10.1.2 Initialize System Data Base (S\$NEW). S\$NEW initializes a data base for use by the various system routines according to the terminal state, mode, and ID. Some command processors do not call S\$GTCA because they do not need to access the TCA; however, these processors must call S\$NEW before they can use any other S\$ routines. Processors that call S\$GTCA need not call S\$NEW.

Calling Sequence

```
BLWP    @S$$NEW
```

Registers Used

R0 — Error code returned by S\$NEW

Error Codes

>FFFF — S\$NEW previously called

3.10.1.3 Bid Task Routine (S\$BIDT). S\$BIDT allows tasks that are normally bid via the .BID or .QBID primitives to be bid from another task. S\$BIDT allows a task to replicate the SCI environment at the time a .BID is performed. This capability is also available in COBOL through C\$BID (which in turn uses S\$BIDT) and is available to other languages through an interface to an assembly language routine.

By using S\$BIDT, a task can execute other user-written tasks or a system task directly, without returning to SCI to have the .BID primitive execute the task. This allows the task to retain control and avoids the problem of having to reenter the task again to resume processing upon return from SCI.

Since S\$BIDT can bid any task that the .BID or .QBID primitives can bid, the routine can invoke tasks that are part of the operating system release as well as user-written tasks. For example, you have an application task that prompts the user for a directory pathname. You could use the S\$BIDT routine to call the List Directory task in order to produce an alphabetized listing of the members in the directory. The application program could then read this alphabetized listing and prompt the user for desired actions on selected members. After interfacing with the user, the application task could use S\$BIDT to call the Copy Directory task to copy specified members to another directory.

Use of the S\$BIDT routine does, however, have several drawbacks:

- To call a task directly with S\$BIDT routine means bypassing the command procedure. Therefore, you risk the fact that a call to a released operating system task may not work on a later release of the operating system due to a change in the .BID calling sequence.
- The task using S\$BIDT must be invoked via a .BID, .QBID, or .DBID primitive. However, a task bid via S\$BIDT can issue an S\$BIDT call to enable concatenation. If the task using S\$BIDT is invoked via XT (Execute Task) or XTS (Execute Task and Suspend SCI), the results can be unpredictable.
- If the task calling S\$BIDT has opened the terminal local file (TLF), the task bid by the S\$BIDT routine will be unable to get access to the TLF and may terminate without performing the desired action. You can get around this problem, however, by having one of the tasks use an alternate pathname. You can either use DUMMY or define your own pathname.

- If the S\$BIDT routine encounters errors when bidding the task, it returns error code. However, any error code returned by a task that is called via the S\$BIDT routine is very difficult to access. If one of your application programs absolutely needs to access this error code, contact your Customer Representative for information.

As with all S\$ routines, a task must call either S\$GTCA or S\$NEW (to perform the necessary initializations) before it calls S\$BIDT.

Calling Sequence

```
BLWP @S$BIDT
```

Registers Used

- R0 — Set to 0 by S\$BIDT if the task was bid successfully. Set to a nonzero error code by S\$BIDT if the task was not bid successfully.
- R1 — The calling task sets the MSB to the task ID of the task to be executed and the LSB to the LUNO assigned to the program file of the task to be executed.
- R2 — Set by the calling task to contain the address of an address table. Each of the addresses in the table points to a parameter that will be passed to the task to be executed. This address table must contain a 0 as its last entry. If R2 itself is 0, the parameters of the calling task are passed to the task to be executed.
- R3 — The calling task sets the MSB to the CODE to be passed to the task to be executed. The MSB has an integer range between 0 and 255. It is 0 if no CODE is to be passed; all other values correspond to the CODE keyword which may optionally appear on the .BID statement. The calling task uses the LSB to set flag bits:
- Bit 8 — If set to 1, the run ID of the bid task is returned in the MSB of R1. If set to 0, R1 remains unchanged.
 - Bit 9 — If set to 0, the calling task is associated with the same station as the caller. If set to 1, the task is not associated with a station.
 - Bit 10 — If set to 1, the task shares the synonym table with the caller. If set to 0, the called task gets a snapshot of the current synonyms in a table of its own.
 - Bit 11 — Set to 0.
 - Bit 12 — If set to 1, the calling task is terminated after the called task is bid.
 - Bit 13 — Set to 0.
 - Bit 14 — Set to 0.
 - Bit 15 — If set to 1, the calling task is suspended until the called task is terminated.

If neither bit 12 or bit 15 of R3 is set to 1, the called task and the calling task will run concurrently. Therefore, you must not expect the function of the called task to finish at any particular time.

Error Codes

> FFFD — S\$BIDT is unable to make a snapshot of the current synonyms.

> FFFE — S\$BIDT is unable to bid the task specified.

S\$BIDT can return any error returned by an S\$ routine. You can refer to the SCI section of the *DNOS Messages and Codes Manual* for help.

To illustrate the use of the S\$BIDT routine, the following example shows you how to have S\$BIDT call the Copy/Concatenate (CC) processor:

1. Examine the command procedure to determine the input values needed for the S\$BIDT routine.

```
CC(COPY/CONCATENATE),
INPUT ACCESS NAME(S)   =(ACNM)("@$$F$P"),
OUTPUT ACCESS NAME     =ACNM
REPLACE?               =ELEMENT(Y=YES,N=NO)(NO),
MAXIMUM RECORD LENGTH =*INT
.SYN $$F$P="&INPUT ACCESS NAME"
*BID TASK CCAF
.BID TASK=011, UTILITY, CODE=1,
PARMS=((&INPUT ACCESS NAME),"&OUTPUT ACCESS NAME",
NO,"&REPLACE",NO,"&MAXIMUM RECORD LENGTH")
```

2. Set up the proper data structure to invoke the command procedure. Notice that the CC command procedure prompts for responses. It then bids task > 11 with a code value of 1 and passes six parameters. Under DNOS 1.2, you could invoke the CC procedure with the following data structure:

```
ADRTBL DATA INPUT
        DATA OUTPUT
        DATA PARM3
        DATA REPLAC
        DATA PARM5
        DATA MAX
        DATA 0

INPUT  BYTE 5
      TEXT (.IN)
OUTPUT BYTE 4
      TEXT .OUT
PARM3  BYTE 2
      TEXT NO
REPLAC BYTE 1
      TEXT Y
PARM5  BYTE 2
      TEXT NO
MAX    BYTE 0
```

Notice how the parameters being passed match up with the six parameters of the CC command procedure. The first parameter passed to the CC procedure is a five-byte string “(.IN)”, which represents the input file. The second parameter is a four-byte string “.OUT”, which represents the output file. The third parameter is always “NO”. The fourth parameter is a one-byte string “Y”, which indicates the output file is to be replaced if one already exists. The fifth parameter is always “NO”. The sixth parameter, maximum record length, is not being used. Notice also how ADRTBL is terminated by a 0 and has parameters with the following format: byte 0 is equal to the number of characters in the parameter; bytes 1 through N contain the ASCII characters of the parameter.

3. Set up code to execute the command procedure. Assuming you are using DNOS 1.2 and the calling task is on S\$UTIL, you could bid the CC task with the following code:

```

LI      R1,>11FF          TASK ID >11, LUNO FF
LI      R2,ADRTBL        ADDRESS OF TABLE
LI      R3,>0101         CODE >01, SUSPEND CALLER
BLWP   @S$BIDT          BID CC UTILITY TASK

```

This code sets up the registers with the values needed by the CC utility and then passes control to the S\$BIDT routine. S\$BIDT collects the input parameters, performs other processing necessary to replicate the SCI environment for the bid, and bids the CC task with a copy of the caller’s synonyms and logical names.

By following the general principles of this example, you can use the S\$BIDT routine to execute other DNOS or user-written tasks.

3.10.1.4 Get Parameter (S\$PARM). S\$PARM returns the parameters in the TCA to the command processor. These are the PARMs parameters of the .BID, .QBID, or .DBID primitive. These parameters are text strings separated by commas. Place an integer indicating the number of the parameter desired in register Ra. Place the address of a buffer into which the text string is to be copied in register Rb. If the buffer is too short, an error code is returned in register R0. Registers Ra and Rb are specified in the two bytes immediately following the call to S\$PARM.

Calling Sequence

```

BLWP   @S$PARM
BYTE   Ra,Rb

```

Registers Used

- R0 — Error code returned by S\$PARM
- Ra — Index of desired parameter
- Rb — Address of a buffer in which to place the parameter text string

Error Codes

- > 901B — Output buffer too small
- > FF05 — Unable to access NCT

Example

This example retrieves the second parameter text string. The .BID primitive for task EXAM is as follows:

```
.BID TASK = EXAM, CODE = 37, PROG = VOL.TEST.PROG,
      PARS = ("&NAME", "@OLDVAL"))
```

The command processor accesses the parameter as follows:

```
BUF      BYTE 20
         BSS 20
         EVEN

         REF      $$GTCA, $$PARM, $$RTCA, $$TERM
         :
         :
         :
BLWP     @$GTCA          GET TCA
MOV      R0,R0          CHECK FOR ERRORS
JNE      ERROR
LI       R4,2           ESTABLISH PARAM INDEX
LI       R5,BUF         ESTABLISH BUFFER POINTER
BLWP     @$PARM         GET 2ND PARM
BYTE    R4,R5          $$PARM INFO
MOV      R0,R0          CHECK FOR ERRORS
JNE      ERROR
BLWP     @$RTCA         RELEASE TCA
MOV      R0,R0          CHECK FOR ERRORS
JNE      ERROR
         :
         :
ERROR    LI       R1,>C000  CONDITION CODE=C000
         CLR      R2          NO VARIABLE TEXT
         LI       R3,2       SCI ERROR
         MOV      R0,R4      MESSAGE NUMBER
         BLWP     @$TERM     TERMINATE TASK
```

3.10.1.5 Get Terminal Status (\$\$STAT). \$\$STAT returns the status of the terminal from which the command processor was activated. The information is returned as a 32-bit integer. The first byte contains the following:

Bit	Contents																
0	Reserved																
1 – 3	User privilege code. The hexadecimal values and privilege levels are as follows: <table border="0" style="margin-left: 40px;"> <tr><td>0</td><td>Lowest level of access</td></tr> <tr><td>1</td><td>User defined</td></tr> <tr><td>2</td><td>System access level</td></tr> <tr><td>3</td><td>User defined</td></tr> <tr><td>4</td><td>Management access level</td></tr> <tr><td>5</td><td>User defined</td></tr> <tr><td>6</td><td>System and management access level</td></tr> <tr><td>7</td><td>User defined</td></tr> </table>	0	Lowest level of access	1	User defined	2	System access level	3	User defined	4	Management access level	5	User defined	6	System and management access level	7	User defined
0	Lowest level of access																
1	User defined																
2	System access level																
3	User defined																
4	Management access level																
5	User defined																
6	System and management access level																
7	User defined																
4 – 7	Current terminal mode, in the form of a hexadecimal number: <table border="0" style="margin-left: 40px;"> <tr><td>0</td><td>Batch mode or background</td></tr> <tr><td>1</td><td>TTY mode</td></tr> <tr><td>F</td><td>VDT mode</td></tr> </table>	0	Batch mode or background	1	TTY mode	F	VDT mode										
0	Batch mode or background																
1	TTY mode																
F	VDT mode																

The second byte contains the station ID, the third byte is reserved, and the fourth byte contains the CODE value of the most recent .BID, .DBID, or .QBID.

Calling Sequence

`BLWP @$STAT`

Registers Used

- R0 — Error code returned by \$\$STAT
- R3 — Address of 32-bit buffer

Error Codes

None (currently)

You can issue a call to \$\$STAT prior to a call to \$\$GTCA; also, you can issue a call to it after calling \$\$RTCA if you call \$\$NEW first.

3.10.1.6 Set Synonym Value (\$\$SETS). The \$\$SETS routine defines or redefines a synonym in the NCT. Place the synonym in a text string buffer at the address in register Ra. Place the value to be assigned the synonym in a buffer at the address in register Rb. If register Rb contains 0 or the address of a zero-length string, the synonym is deleted from the TCA.

Calling Sequence

```
BLWP  @S$SETS
BYTE  Ra,Rb
```

Registers Used

R0 — Error code returned by S\$SETS
 Ra — Address of synonym name text string
 Rb — Address of synonym value text string

Error Codes

> FF05 — Unable to access NCT
 > FF06 — Synonym table overflow

Example

```
SYNAME BYTE 5
        TEXT 'SYN01'
VALUE  BYTE 14
        TEXT 'DS02.LIB.INPUT'
        .
        .
        .
        LI   R3,SYNAME
        LI   R4,VALUE
        BLWP @S$SETS
        BYTE R3,R4
        MOV  R0,R0
        JNE  ERROR
```

3.10.1.7 Get Synonym Value (S\$MAPS). S\$MAPS searches the NCT for the synonym name in a buffer at the address in register Ra. With this routine, you can access only synonyms defined by S\$SETS or by the .SYN primitive. When the synonym is found and the output buffer is large enough, the value is placed in the buffer at the address in register Rb. If the buffer is too small, an error code is returned in register R0. If the synonym is not found in the TCA, a zero-length string is copied into the buffer. When the synonym name contains a period (.), the text preceding the period is replaced by its synonym value, if one exists.

Calling Sequence

```
BLWP  @S$MAPS
BYTE  Ra,Rb
```

Registers Used

- R0 — Error code returned by S\$MAPS
- Ra — Address of synonym name text string buffer
- Rb — Address of synonym value text string buffer

Error Codes

- > 901B — Output buffer too small
- > FF05 — Unable to access NCT

Example

```

SYNAME  BYTE 4
        TEXT 'SYNM'
VALUE   BYTE 40
        BSS 40
        .
        .
        LI  R2,SYNAME      Ra=2
        LI  R3,VALUE      Rb=3
        BLWP @S$MAPS
        BYTE R2,R3
        MOV R0,R0
        JNE ERROR
    
```

3.10.1.8 Search Name Correspondence Table (S\$SNCT). S\$SNCT searches the NCT for the synonym that is right before or after the character string at the address in register Ra. The NCT in the TCA contains synonyms. The value in register R0 determines whether the search is for the preceding or the succeeding character in the ASCII character sequence. The original string need not appear in the table.

When the desired synonym is found, the synonym is placed in the buffer at the address in register Ra. Its value may be placed in the buffer at the address in register Rb. If no synonym is found, a null string (length 0) is placed in the buffer at the address in register Ra. When Ra contains the address of a zero-length string and R0 contains 0, the routine returns the first synonym in ASCII code order. When Ra contains the address of a zero-length string and R0 contains - 1, the routine returns the last synonym in ASCII code order. When register Rb contains 0, no value is returned. The synonym is returned in the buffer at the address in register Ra. Use S\$SNCT to access synonyms in ASCII code order.

\$\$\$NCT assumes the following:

- The Ra and Rb buffers are 255 bytes long (if Rb is not 0).
- The character count value in the first byte of each buffer is a count of the string currently in the buffer. Unless Rb is 0, **\$\$\$NCT** does not check the buffer length before writing in the buffer.

Calling Sequence

```
BLWP @$$$NCT
BYTE Ra,Rb
```

Registers Used

R0 — Set to 0 to search for successor and - 1 to search for predecessor
 Ra — Address of buffer containing original string; synonym name is returned here
 Rb — Pointer to the buffer that receives either the value of the synonym found or 0

Error Codes

None (currently)

Example

```

SYN          BYTE 3          LENGTH OF SYN BUF
NAME        TEXT 'SYN'
           BSS 252
VALUE      BYTE 255        LENGTH OF VALUE BUF
           BSS 255
           .
           .
           .
           LI R0,0          R0 → GET SUCCESSOR
           LI R3,SYN        R3 → NAME OF SYN
           LI R4,VALUE      R4=VALUE BUFFER
GETNXT      BLWP @$$$NCT    GET NEXT SYN
           BYTE R3,R4      DEFINE 'A' AND 'B'
           MOVB *R3,R1     MOVE R0,R0 IN ERROR
           JEQ OUT         CHECK FOR END OF NCT
*PROCESS THE SYNONYM
```

3.10.1.9 Split List Into Components (\$\$SPLT). **\$\$SPLT** divides a list and returns the first element and the remainder of the list separately. **\$\$SPLT** copies the first element (all text that precedes the first comma of the list) in the buffer at the address in register R1 into the buffer at the address in register R2. The routine also copies the remainder of the list into the buffer at the address in register R3. Registers R1 and R3 can contain the same address.

Calling Sequence

BLWP @S\$SPLT

Registers Used

- R0 — Error code returned by S\$SPLT
- R1 — Address of list text string
- R2 — Address of buffer to receive first element of list
- R3 — Address of buffer to receive remainder of list

Error Codes

- >901B — Output buffer too small
- >FFFF — Unbalanced parentheses

Example

```

LIST      BYTE 33                LENGTH OF LIST
          TEXT '(20,LIST ACCESS'
          TEXT 'NAME,OUTPUT FILE)'
FIRST    BYTE 20                LENGTH OF 'FIRST' BUF
          BSS 20
          .
          .
          .
          LI  R1,LIST            R1 → LIST POINTER
          LI  R2,FIRST          R2 → FIRST POINTER
          MOV R1,R3             R3 → REST POINTER =R1
          BLWP @S$SPLT
          MOV R0,R0
          JNE ERROR
    
```

3.10.1.10 Return Time and Date (S\$TAD). S\$TAD returns the time and date information that DNOS maintains. The routine issues an SVC to obtain the date and time block. The date and time from the block are formatted and returned to the calling task. For an initialized date, the string has the following form:

HR:MIN:SEC WEEKDAY, MONTH, DAY, YEAR.

When the time and date have not been initialized, only the time is returned. The values returned represent the elapsed time since power was applied to the computer.

Calling Sequence

BLWP @S\$TAD

Registers Used

- R0 — Error code returned by S\$TAD
- R1 — Address of buffer for time and date

Error Codes

> 901B — Output buffer too small

Example

This example shows a string returned by S\$TAD:

14:48:16 FRIDAY, NOV 07, 1980.

3.10.1.11 Put TCA (S\$PTCA). S\$PTCA should be called before the processor terminates or calls S\$RTCA. This routine is provided for compatibility with other Model 990 Computer operating systems and for future DNOS development.

Calling Sequence

BLWP @S\$PTCA

Registers Used

R0 — Error returned by S\$PTCA

Error Codes

None (currently)

3.10.1.12 Release TCA (S\$RTCA). index(S\$RTCA Routine) S\$RTCA releases the TCA. The command processor should call it just before terminating. This routine is provided for compatibility with other Model 990 Computer operating systems and for future DNOS development.

Calling Sequence

BLWP @S\$RTCA

Registers Used

R0 — Error code returned by S\$RTCA

Error Codes

None (currently)

3.10.1.13 Create Message (S\$CMMSG). S\$CMMSG writes a message in a buffer using information supplied in registers defined in the calling sequence. Use routine S\$CMMSG to return an error or status message when the command processor continues processing after issuing the message. Use routine S\$TERM to issue an error or status message and terminate the task.

Calling Sequence

BLWP S\$CMMSG

Registers Used

- R0 — Error code returned by S\$CMMSG.
- R1 — Address of the buffer in which the message is returned. The address must be a word-aligned (even) address. (The first full word contains the buffer length in bytes.)
- R2 — Address of buffer that contains the variable text.
- R3 — Error source information.
- R4 — Internal message code.
- R5 — Address of buffer that contains either the final component of the message file pathname or 0.
- R6 — Address of buffer that contains additional variable text; applies only when bit 4 of error source information word is set to 1.
- R7 — Extra flags word, if R3 so indicates.

Error Codes

- > 901B — Output buffer is too small

Calling this routine requires some analysis of the error condition prior to the call, specifically the following:

- For an SVC error, execute a Return Code Processing SVC (opcode > 4C) to obtain the message number and other required data.
- Obtain the file name component of the message file pathname by accessing the value of a synonym. You can also obtain the file name if the message file is a system message file with a known file name or if the message file is specified with a file indicator.

Register R1 contains the address of a buffer into which the routine places the message. A message can be more than 255 characters long; the first word of the buffer contains the length of the text portion (the remainder) of the buffer. The address must be a word-aligned address. The buffer must allow enough space for the message from the file and any anticipated variable text.

Set register R2 to one of the following:

- The address of a buffer that contains the variable text for the message
- The address of a zero-length buffer when no variable text is required
- Zero

The count includes the semicolons (;) that separate the variable text elements. A message can contain as many as nine variable text elements. You must place data in the buffer that corresponds to the variable text elements defined for the message.

Register R3 contains error source information consisting of four hexadecimal digits. In most cases, register R3 is set to 0, indicating that all error source information comes from the file. To override the file information, set the leftmost hexadecimal digit to specify the error source as follows:

Value	Meaning
0	Use source information in error file
1	Warning
2	User error
4	System error
6	User or system error
8	Hardware error
A	User or hardware error
C	System or hardware error
E	User, system, or hardware error
F	Informative message

Set the second digit to 0 when no additional variable text is required; set it to 8 to place additional text at the end of the message. When you set this digit to 8, you should place the address of the buffer that contains this text in register R6. To suppress header information, add 1 to the second digit (making it 9 or 1). Set register R7 to >8000 to indicate header suppression. A 1 in the second digit of R3 indicates that register R7 is an additional flags word.

Set the third and fourth digits to the file indicator of the message file, as follows:

Setting	Meaning
>00	No message file or file identified in register R5
>01	SVC error message file
>02	Utility error message file
>xy	Last component of the pathname to which synonym \$\$\$\$FNxy is assigned (where xy is greater than or equal to >80)

Set register R4 to the internal message code of the desired message.

Set register R5 to the address of a buffer that contains the file name (last) component of the file pathname of the message file that contains the message or that is set to 0. The first byte of the buffer must contain the number of bytes of the file name component. This register should be set to 0 if the file indicator in R3 identifies the message file or if no message file is specified. When the file indicator is 0 and register R5 contains either 0 or the address of a null string, an abbreviated message is written. The abbreviated message consists of the file indicator, message number, and variable text.

Set register R6 to the address of a buffer that contains additional variable text. Set it to 0 when no additional text is required. When the second digit of the value in R3 is not set to 8, register R6 does not apply. Additional variable text is placed at the end of the specified message independently of any question marks in the file message.

If bit 7 of register R3 equals 1, register R7 is used as an additional flags word. If bit 0 of register R7 equals 1, message headers are suppressed.

Section 9 provides further information on message file format.

3.10.1.14 Terminate and Return to SCI (S\$TERM). S\$TERM sets the termination synonyms and terminates the calling task. The following are the termination synonyms:

Synonym	Meaning
\$\$CC	Condition code
\$\$VT	Variable text
\$\$ES	Error source
\$\$MN	Internal message code
\$\$FN	Message file name

Calling Sequence

BLWP @S\$TERM

Registers Used

- R1 — Value for condition code \$\$CC
- R2 — Address of buffer that contains variable text
- R3 — Error source information
- R4 — One of the following:
 - 0 (normal termination)
 - Internal message number (non-SVC-detected error)
 - Address of SVC block (SVC-detected error to report)

SCI uses the values of the termination synonyms to provide a warning or error message. When you call routine S\$TERM at successful termination of a command processor, set registers R1 through R4 to 0 to terminate without issuing a message.

The condition code synonym contains the severity code. Set register R1 to one of the following values:

Value	Meaning
> 4000	Terminated with a warning message
> 8000	Terminated with an error message for a recoverable error
> C000	Terminated with an error message for a fatal (unrecoverable error)

Set register R2 to the address of a buffer that contains the variable text for the message. Set it to 0 when no variable text is required. The count of characters in the buffer includes the semicolons (;) that separate the variable text elements. A message can contain as many as 9 variable text elements but no more than 235 characters of text. Data must be placed in the buffer that corresponds to the variable text elements defined for the message.

Register R3 contains error source information consisting of four hexadecimal digits. In most cases, register R3 is set to 0, indicating that all error source information comes from the file. If you need to override the file information, set the most significant digit to specify the error source as follows:

Value	Meaning
0	Use source information in error file
1	Warning
2	User error
4	System error
6	User or system error
8	Hardware error
A	User or hardware error
C	System or hardware error
E	User, system, or hardware error
F	Informative message

Set the second digit to 0. Set the third and fourth digits to the file indicator of the message file, as follows:

Setting	Meaning
> 00	No message file
> 01	SVC error message file
> 02	Utility error message file
> xy	Last component of the pathame to which synonym \$\$FNxy is assigned

Set register R4 to the message number unless the file indicator in register R3 is > 01 (SVC error). For an SVC error, set R4 to the address of the SVC block.

3.10.1.15 Alternate Termination (\$\$STOP). \$\$STOP is included in DNOS to support command processors for earlier operating systems. \$\$STOP terminates a command processor and returns control to SCI. Routine \$\$TERM performs a similar function with the added capability of providing error messages in DNOS format. Use \$\$TERM in any command processors you write.

3.10.2 Local Display File Routines

The TLF is a file of ASCII data to be displayed. Use the TLF for short messages and listings. SCI provides a TLF for foreground, background, and batch job modes. SCI displays the contents of the foreground TLF immediately prior to displaying the command prompt. The contents of the background TLF appear on the screen when you enter either a WAIT or a Show Background Status (SBS) command. SCI copies the batch mode TLF into the batch listing file. After displaying the TLF, SCI deletes each message.

The routines described in the following paragraphs open files, close files, and build and write records to the file. The maximum length of a TLF record is 134 characters. Data items are written at specific columns, and each line is terminated by a call to \$\$WEOL. When the text is directed to a device instead of to a file, these routines add the required device control characters to the text. Only tasks executed by means of the .BID, .QBID, or .DBID primitives can successfully call these routines.

3.10.2.1 Open File (S\$OPEN). S\$OPEN opens the TLF or a user-specified file for write access. If register R1 contains 0, S\$OPEN opens the TLF.

Calling Sequence

BLWP @S\$OPEN

Registers Used

- R0 — Error code returned by S\$OPEN
- R1 — 0 or the address of the buffer that contains the pathname of the device to call or file to open

Error Codes

- > A1xx — Assign or open error, I/O error code xx
- > 9022 — Invalid use of device
- > 9026 — Invalid file type

3.10.2.2 Open File Specifying User ID (S\$OPNS). S\$OPNS opens a specified file in the same way S\$OPEN does but has one additional feature: when the Assign LUNO is performed on the file, a specified user ID and passcode are used for security purposes. However, if the calling task is a security bypass task, the passcode field is ignored.

If register R1 contains 0, S\$OPNS opens the TLF.

Calling Sequence

BLWP @S\$OPNS

Registers Used

- R0 — Error code returned by S\$OPNS.
- R1 — 0 or the address of the buffer that contains the pathname of the device or file to open
- R2 — Address of a buffer with the user ID parameters. The buffer begins with two bytes values. The first byte has a value of > 02; the second a value of > 10. These bytes are then followed by two eight-character fields. The first contains the user ID; the second the password. Each of these two fields should be right filled with blanks if the values are less than eight characters long.

Error Codes

- > A1xx — Assign or open error, I/O error code xx
- > 9022 — Invalid use of device
- > 9026 — Invalid file type

3.10.2.3 Write to File (S\$WRIT). S\$WRIT concatenates the text string addressed by register R1 with the current line to be written to the file. When register R2 contains 0 or a positive value, the value specifies the column (0 through 133) in which the text begins. A negative value in R2 is not valid. When any byte in the string contains >7F, the immediately preceding character is repeated. The byte following the byte that contains >7F specifies the number of repetitions. The string should not contain device control characters, such as a line feed; S\$WRIT supplies these as needed.

Calling Sequence

```
BLWP @S$WRIT
```

Registers Used

R0 — Error code returned by S\$WRIT
 R1 — Address of text to be written
 R2 — Starting column in the record

Error Codes

> FFF8 — File is not open
 > FFF9 — Start position is too small
 > FFFA — Text buffer overflow

3.10.2.4 Write End-of-Line to File(S\$WEOL). S\$WEOL terminates the current line to be written and writes it to the file. If S\$WRIT has not supplied any text since the file was opened or since the previous line was written, S\$WEOL writes a blank line.

Calling Sequence

```
BLWP @S$WEOL
```

Registers Used

R0 — Error code returned by S\$WEOL

Error Codes

> A1xx — I/O error xx has occurred
 > FFF8 — File is not open

3.10.2.5 Close File (S\$CLOS). S\$CLOS terminates writing to the file. You should call S\$CLOS if the file was opened prior to a call to S\$TERM. When register R1 contains 0 and the file is the TLF, the TLF appears on the screen after the command completes and before the termination message appears.

When R1 contains a nonzero value, the lines that were written to the TLF since the last call to S\$OPEN or S\$OPNS are erased from the TLF.

Calling Sequence

BLWP @S\$CLOS

Registers Used

R0 — Error code returned by S\$CLOS
 R1 — Display flag

Error Codes

> FFF8 — TLF is not open

3.10.2.6 Local Display File Example. The following example includes a call to each of the local display file routines:

Example

```

M1      BYTE  10
        TEXT  'THIS IS A'
M2      BYTE  16
        TEXT  'TLF TEST MESSAGE'
        .
        .
        .
        CLR   R1           TLF TO BE OPENED
        BLWP  @S$OPEN     OPEN TLF
        MOV   R0,R0       TEST FOR ERROR
        JNE   ERROR
        .
        .
        .
        LI   R1,M1        MESSAGE ADDRESS
        LI   R2,0         COLUMN ADDRESS
        BLWP @S$WRIT     WRITE M1
        MOV   R0,R0       TEST FOR ERROR
        JNE   ERROR
        .
        .
        .
        LI   R1,M2        MESSAGE ADDRESS
        LI   R2,11        COLUMN ADDRESS
        BLWP @S$WRIT     WRITE M2
        MOV   R0,R0       TEST FOR ERROR
        JNE   ERROR
        .
        .
        .
    
```

```

        BLWP    @S$WEOL    WRITE END-OF-LINE
        MOV     R0,R0      TEST FOR ERROR
        JNE     ERROR
        CLR     R1         CLEAR DISPLAY FLAG
        BLWP    @S$CLOS    CLOSE & DISPLAY TLF
        MOV     R0,R0      TEST FOR ERROR
        JNE     ERROR
        .
        .
        .
ERROR   LI      R1,>C000   CONDITION CODE=C000
        CLR     R2         NO VARIABLE TEXT
        LI      R3,2      SCI ERROR
        MOV     R0,R4     MESSAGE NUMBER
        BLWP    @S$TERM   TERMINATE TASK

```

3.10.3 String Utility Routines

The string utility routines copy, compare, and convert character strings. The string buffer required by these routines has the form previously described.

An empty buffer reserved for string storage should indicate the size of the buffer (minus 1) in the first byte.

The string utility routines are as follows:

- S\$INT — Convert ASCII to Binary Integer
- S\$IASC — Convert Binary Integer to ASCII
- S\$SCOM — Compare Strings
- S\$SCPY — Copy String

3.10.3.1 Convert ASCII to Binary Integer (S\$INT). S\$INT converts an ASCII text string that represents an integer expression into a 32-bit binary value. The integer expression to be converted can contain the standard arithmetic operators +, -, *, and /. Register R4 contains the base of the numbers to be converted. When the ASCII string contains numbers beginning with > or 0, the numbers are converted as hexadecimal numbers regardless of the base specified in register R4.

Calling Sequence

```
BLWP @S$INT
```

Registers Used

- R0 — Error code returned by S\$INT
- R2 — Address of buffer that contains ASCII code to be converted to binary integer
- R3 — Address of a 32-bit buffer in which the converted value is stored
- R4 — Base of number represented by the input string; a 0 value indicates base 10

Error Codes

- > 9002 — Invalid integer expression
- > FFFF — Divide by zero

Example

BUFF	BYTE	LNG	TEXT STRING TO BE CONVERTED
	TEXT	'33000'	
LNG	EQU	\$-BUFF-1	LENGTH CALCULATION
NMB	BSS	4	BUFFER FOR BINARY VALUE
	.		
	.		
	.		
	LI	R2,BUFF	ADDRESS OF TEXT STRING
	LI	R3,NMB	BUFFER ADDRESS FOR BINARY NO
	LI	R4,0	SET FOR BASE 10
	BLWP	@S\$INT	CONVERT TEXT STRING TO BINARY
	MOV	R0,R0	PASS ERROR CODE
	JNE	ERROR	
	.		
	.		

3.10.3.2 Convert Binary Integer to ASCII (S\$IASC). S\$IASC converts a 32-bit binary integer into an ASCII text string representing that number. The 32-bit integer is converted as a two's complement number or a 32-bit positive binary number, depending on the base specified in register R3. If the base is 0, the number is converted as a two's complement binary integer. It is converted into the ASCII representation of the decimal (base 10) number with leading blanks and a minus sign for a negative number. If the base does not equal 0, the 32-bit integer is considered to be positive; it is converted into the ASCII representation of the integer in the specified base, with leading zeros.

Calling Sequence

BLWP @S\$IASC

Registers Used

- R0 — Error code returned by S\$IASC.
- R1 — Address of the 32-bit integer.
- R2 — Address of the buffer to receive the ASCII text string. The first byte of the buffer must contain the buffer length minus 1. The buffer must be large enough to contain the largest possible values.
- R3 — In byte 0, number of ASCII characters to be returned; 0 means variable number; maximum is 32.
In byte 1, base (for example, 10 or 16) into which integer is to be converted prior to representation in ASCII; 0 means decimal.

Error Codes

- >901B — Output buffer is too small
- >FFFF — Field width is greater than 32

Example

```

BUFF      BYTE 15          BUFFER FOR ASCII VALUE
          BSS 15
NMB       DATA >20       NUMBER = >20
          DATA 0
          .
          .
          .
          LI R2,BUFF       ADDRESS OF TEXT STRING
          LI R1,NMB        ADDRESS OF BINARY NUMBER
          LI R3,0          VARIABLE LENGTH/BASE 10
          BLWP @S$IASC     CONVERT BINARY TO ASCII
          MOV R0,R0        PASS ERROR CODE
          JNE ERROR
          .
          .
    
```

3.10.3.3 Compare Strings (S\$SCOM). S\$SCOM compares two strings and sets the equal and arithmetic greater than bits (bits 1 and 2) of the status register and register R0 to the results of the comparison. If one string is shorter than the other, it is treated as if it is filled to the right with null characters (>00). If one string is a substring of the other (matching from the left), R0 is set to 0. The addresses of the two strings are in registers Ra and Rb. Registers Ra and Rb are specified in the two bytes immediately following the call.

Calling Sequence

```

BLWP @S$SCOM
BYTE Ra,Rb
    
```

Registers Used

- R0 — Substring test code returned by S\$SCOM: 0 = one string is a substring – 1 = strings do not match
- Ra — Address of the buffer that contains the first string
- Rb — Address of the buffer that contains the second string

Error Codes

Status returned in R0

Example

```

FIRST      BYTE 6           LENGTH OF FIRST
           TEXT 'SUBSTR'   STRING
SECOND     BYTE 9           LENGTH OF SECOND
           TEXT 'SUBSTRING' STRING
           LI R3,FIRST     R3 POINTS TO FIRST
           LI R5,SECOND    R5 POINTS TO SECOND
           BLWP @S$SCOM    COMPARE THE TWO
           BYTE R3,R5      DEFINE 'A' AND 'B'
           JEQ OUT         THIS JUMP WILL NOT
           MOV R0,R0       OCCUR
           JEQ SUB         THIS JUMP WILL OCCUR
    
```

3.10.3.4 Copy String (S\$SCP Y). S\$SCP Y copies the string at the address in register Ra into the buffer at the address in register Rb, placing the length of the copy string in the first byte of the buffer. Registers Ra and Rb are specified in the two bytes immediately following the call. The buffer containing the string to be copied must not overlap the receiving buffer. If the length of the receiving buffer is less than the length of the string to be copied, an error code is returned in register R0. When register Ra contains 0 or the address of a null string (zero length), the first byte of the buffer at the address in register Rb is set to 0.

Calling Sequence

```

BLWP @S$SCP Y
BYTE Ra,Rb
    
```

Registers Used

- R0 — Error code returned by S\$SCP Y
- Ra — Address of buffer that contains text to be copied
- Rb — Address of buffer to receive copy

Error Codes

>901B — Output buffer too small

Example

```

STRING    BYTE 7           LENGTH OF STRING
           TEXT 'COPY ME'
COPY      BYTE 20          LENGTH OF BUFFER
           BSS 20
           LI R1,STRING    R1=POINTER TO STRING
           LI R8,COPY      R8=POINTER TO BUFFER
           BLWP @S$SCP Y  CALL S$SCP Y
           BYTE R1,R8     DEFINE 'A' AND 'B'
           MOV R0,R0      TEST FOR ERROR
           JNE ERROR
           .
           .
           .
    
```

```

ERROR  LI      R1,>C000  CONDITION CODE=C000
        CLR    R2      NO VARIABLE TEXT
        LI     R3,2     SCI ERROR
        MOV    R0,R4    MESSAGE NUMBER
        BLWP  @S$TERM  TERMINATE TASK

```

3.10.4 Arithmetic Utility Routines

The arithmetic utility routines perform addition, subtraction, multiplication, and division with 32-bit signed integer operands. The operands must be in binary form. All of the routines allow the addresses of the operands and the addresses of the results to be the same. The logical greater than, arithmetic greater than, and equal bits in the status register (bits 0 through 2) are set or reset as assembly language instructions would set them.

The following routines are available:

S\$IADD — Add 32-bit integers

S\$ISUB — Subtract 32-bit integers

S\$IMUL — Multiply 32-bit integers

S\$IDIV — Divide 32-bit integers

3.10.4.1 Add 32-Bit Integers (S\$IADD). S\$IADD adds two 32-bit integers in two's complement form. The sum is a 32-bit two's complement integer. Registers R1 and R2 contain the addresses of the two integers, and the sum is placed in the address in register R3.

Calling Sequence

```
BLWP @S$IADD
```

Registers Used

R0 — Error code returned by S\$IADD

R1 — Address of the 32-bit buffer containing the addend

R2 — Address of the 32-bit buffer containing the addend

R3 — Address of the 32-bit buffer for the sum

Error Codes

> FFFF — Numeric overflow

Example

NUM1	DATA >0000	BUFFER FOR 32-BIT INTEGER
	DATA >1111	
NUM2	DATA 0	BUFFER FOR 32-BIT INTEGER
	DATA >0145	
RESLT	BSS 4	BUFFER FOR 32-BIT SUM
	.	
	.	
	.	
	LI R1,NUM1	ADDRESS OF INTEGER
	LI R2,NUM2	ADDRESS OF INTEGER
	LI R3,RESLT	ADDRESS OF RESULT BUFFER
	BLWP @S\$IADD	PERFORM ADDITION
	MOV R0,R0	PASS ERROR CODE
	JNE ERROR	
	.	
	.	

3.10.4.2 Subtract 32-Bit Integers (S\$ISUB). S\$ISUB subtracts 32-bit integers. If register R1 contains 0, S\$ISUB calculates the negative of the number at the address in register R2, that is, 0 minus the number.

Calling Sequence

BLWP @S\$ISUB

Registers Used

- R0 — Error code returned by S\$ISUB
- R1 — Address of the 32-bit buffer containing the minuend
- R2 — Address of the 32-bit buffer containing the subtrahend
- R3 — Address of the 32-bit buffer for the difference

Error Codes

> FFFF — Numeric overflow

Example

NUM1	DATA >0000	BUFFER FOR 32-BIT INTEGER
	DATA >1111	
NUM2	DATA 0	BUFFER FOR 32-BIT INTEGER
	DATA >0145	
RESLT	BSS 4	BUFFER FOR 32-BIT RESULT
	.	
	.	
	L1 R1,NUM1	ADDRESS OF INTEGER
	L1 R2,NUM2	ADDRESS OF INTEGER
	LI R3,RESLT	ADDRESS OF RESULT BUFFER
	BLWP @S\$ISUB	
	MOV R0,R0	PASS ERROR CODE
	JNE ERROR	
	.	
	.	

3.10.4.3 Multiply 32-Bit Integers (S\$IMUL). S\$IMUL multiplies two 32-bit integers. Registers R1 and R2 contain the addresses of the integers, and S\$IMUL places the 32 least significant bits of the product in the buffer at the address in register R3. No overflow indication is returned.

Calling Sequence

```
BLWP @S$IMUL
```

Registers Used

R0 — Error code returned by S\$IMUL
 R1 — Address of the 32-bit buffer containing the multiplier
 R2 — Address of the 32-bit buffer containing the multiplicand
 R3 — Address of the 32-bit buffer for the product

Error Codes

None (currently)

Example

```

NUM1    DATA >0000          BUFFER FOR 32-BIT INTEGER
        DATA >1111
NUM2    DATA 0              BUFFER FOR 32-BIT INTEGER
        DATA 0145
RESLT   BSS 4                BUFFER FOR 32-BIT INTEGER
        .
        .
        .
        LI R1,NUM1           ADDRESS OF INTEGER
        LI R2,NUM2           ADDRESS OF INTEGER
        LI R3,RESLT          ADDRESS OF RESULT BUFFER
        BLWP @S$IMUL         PERFORM MULTIPLICATION
        MOV R0,R0
        JNE ERROR
        .
        .
    
```

3.10.4.4 Divide 32-Bit Integers (S\$IDIV). S\$IDIV divides the 32-bit integer at the address in register R1 by the 32-bit integer at the address in register R2. The routine places the quotient in the 32-bit buffer at the address in register R3 and the remainder in the 32-bit buffer at the address in register R4. When registers R3 and R4 contain the same address, the quotient is stored at the address and no remainder is stored.

Calling Sequence

```
BLWP @S$IDIV
```

Registers Used

- R0 — Error code returned by S\$IDIV
- R1 — Address of the 32-bit buffer containing the dividend
- R2 — Address of the 32-bit buffer containing the divisor
- R3 — Address of the 32-bit buffer for the quotient
- R4 — Address of the 32-bit buffer for the remainder

Error Codes

> FFFF — Divide by zero

Example

```

NUM1    DATA >0000          BUFFER FOR 32-BIT INTEGER
        DATA >1111
NUM2    DATA 0              BUFFER FOR 32-BIT INTEGER
        DATA >0145
QUOT    BSS 4                BUFFER FOR 32-BIT QUOTIENT
RMDR    BSS 4                BUFFER FOR 32-BIT REMAINDER
        .
        .
        .
        LI R1,NUM1           ADDRESS OF INTEGER
        LI R2,NUM2           ADDRESS OF INTEGER
        LI R3,QUOT           ADDRESS OF QUOTIENT BUFFER
        LI R4,RMDR           ADDRESS OF REMAINDER BUFFER
        BLWP @$$IDIV         PERFORM DIVISION
        MOV R0,R0            PASS ERROR CODE
        JNE ERROR
        .
        .

```

3.10.5 Spooler Interface Routine (\$\$SPLR)

The \$\$SPLR routine allows you to access the spooler subsystem from your task environment. This routine supports all spooler commands that SCI supports. It can be called only from assembly language routines.

Routine \$\$SPLR builds a print request message to the spooler subsystem from information you provide.

Calling Sequence

```
BLWP @$$SPLR
```

Registers Used

R0 — Error code (returned)
R1 — Address of Spooler Control Block (SCB)

Note that R1 is changed to point to an SVC block if error < 90FF is returned in R0.

The SCB template is in the template directory with the other system templates. It contains the following information:

Offset (in Bytes)	Data Type	Contents
0 SCBOP	Byte	Spooler message code
1 SCBFL0	Boolean	Flags
2 SCBFLG	Boolean	Informative flags
4 SCBSID	Character	Spooler ID
10 SCBDEV	Character	Device or class name
18 SCBUSR	Pointer	SCI string for user ID
20 SCBJNM	Pointer	SCI string for job name
22 SCBPTH	Pointer	SCI string for file pathname
24 SCBFRM	Pointer	SCI string for requested form
26 SCBPAG	Word	Integer, number of pages on resume
28 SCBCOP	Byte	Number of copies
29 SCBLPP	Byte	Lines per page
30 SCBPRI	Byte	Job priority
31 SCBXXX	Byte	Reserved
32 SCBDVP	Pointer	SCI string for device or class name

The following values are valid for the SCBOP field:

Value	Meaning
1	Print file message
2	Halt output message
3	Resume output message
4	Kill output message
5	Modify output message
6	Modify Spooler device message
7	Verify validity of device or class name message

The following are SCBFLO field definitions:

Bit	Name	Meaning
0	SCFDVP	True; use SCBDVP rather than SCBDEV
1 through 7		Each bit is reserved and must be 0

The following are SCBFLG field definitions:

Bit	Name	Meaning
0	SCFUSE	True; delete the spooler device
1	SCFAVL	True; not available to the spooler
2	SCFPGD	True; reverse paging on resume output
3	SCFR1	Reserved; must be 0
4	SCFR2	Reserved; must be 0
5	SCFANS	True; ANSI file
6	SCFBNR	True; no banner sheet desired
7	SCFDAP	True; delete after printing
8	SCFIMM	True; halt immediately, not at end-of-file (EOF)
9	SCFR3	Reserved; must be 0
10	SCFR4	Reserved; must be 0
11	SCFR5	Reserved; must be 0
12	SCFSHR	True; remote/shared device
13	SCFDAL	True; delete always (even if a kill output is done later)
14		Reserved; must be 0
15		Reserved; must be 0

S\$SPLR makes the following assumptions about the SCB.

- The device or class name entry must be left justified and blank filled to the right.
- The SCBUSR, SCBPTH, SCBJNM, and SCBFRM fields are SCI string pointers.
- SCBJNM and SCBUSR fields are handled in a special manner. S\$SPLR uses the job name and user ID that the job manager SVC > 48 (Get Job Information) block returns if the field is 0, the length of the string is 0, or the length of the string exceeds eight characters. Otherwise, S\$SPLR uses the user-supplied string.
- When the SCBOP field specifies a print file message, S\$SPLR uses the default form name STANDARD if the SCBFRM field is 0, the length of the string is 0, or the length of the string exceeds eight characters.

- The SCBSID field is returned to you if the print file message is successfully processed.
- If the SCBCOP field is 0 or greater than 127, a value of 1 is used.
- You can send messages using the Modify Spool Device (MSD) command to the spooler, but \$\$\$PLR does not change class name definitions. Also, you cannot specify class names in defining any new device to the spooler.
- If you want to use the Modify Output function, a value of > FF in the SCBPRI and SCBCPY fields indicates that the previous value does not change.

Error Codes

- 0 — Successful completion
- > 90FF — returned in R1
- > 9110 — Invalid device name or class name
- > 906B — Invalid spooler message code (error in \$\$\$PLR)
- > 910F — Spooler cannot assign spool ID to requested file
- > 9112 — Invalid pathname received
- > 9187 — Access is not appropriate to honor request
- > 9194 — Device is not available now
- > 9195 — Device cannot be remote/shared and available to spooler simultaneously
- > 925F — Invalid concatenated pathname syntax
- > 9205 — Number of characters in concatenate pathname set exceeds decimal 256
- > 9207 — Invalid calling sequence to \$\$\$PLR
- > 9206 — SCB aligned on an odd-address boundary
- > 9255 — Invalid spooler ID sent to spooler
- > 907D — Invalid priority specified
- > 9191 — Device is active; request cannot be honored
- > 9193 — Maximum allowable number of devices has been entered
- > 94FF — Errors returned by the job manager SVC > 48, and the I/O subsystem for assign, open, write, close, and release LUNO SVC requests

If an error occurred on an open, write, or close of the spooler channel .\$\$DSTCHN, you must either close the LUNO, release the LUNO, or both. Since the DNOS error processor SVC expects the SVC block causing the error to be intact, no intermediate operations could have been performed using the call block that caused the error. Therefore, \$\$\$PLR cannot close or release the LUNOs, and you must perform these operations after calling the return code processor. If you choose to terminate on the error condition, the operating system closes or releases the LUNOs automatically.

3.10.6 Operator Interface Routines

These routines allow user-written tasks to create operator messages and to receive responses made by the operator.

The operator interface routines are as follows:

- OI\$BGN — Initializes operator interface subsystem
- OI\$COM — Creates an operator message

- OI\$WAT — Causes the task to wait for an operator response
- OI\$END — Terminates the operator interface subsystem session

3.10.6.1 Initialize Operator Interface (OI\$BGN). Routine OI\$BGN must be called before the operator interface is available for use by the calling task. It assigns a LUNO to the operator interface IPC channel.

Calling Sequence

```
BLWP @OI$BGN
```

3.10.6.2 Create Operator Message (OI\$COM). Routine OI\$COM initiates an operator message and returns immediately to the calling task.

Calling sequence

```
BLWP @OI$COM
```

Registers Used

- R1 — MSB: time-out in minutes LSB: number of prompts (0 – 2)
- R2 — Address of buffer containing operator message
- R3 — Address of buffer containing first prompt (if any)
- R4 — Address of buffer containing default response for first prompt (if any)
- R5 — Address of buffer containing second prompt (if any)
- R6 — Address of buffer containing default response for second prompt (if any)

Error Codes

- > 90FF — SVC error
- > 9100 — Number of prompts is greater than 2
- > 9101 — Address pointer of operator response is 0
- > 9102 — Operator message length is 0
- > 9103 — Address pointer of first prompt and default is 0
- > 9104 — Illegal operator message length
- > 9105 — Address pointer of second prompt and default is 0
- > 9106 — Prompt has illegal message length
- > 910C — OI\$COM called previously with reply outstanding
- > 910F — Time delay exceeded

A 0 time-out value specified in the lower byte of register R1 indicates that no response is required. A second prompt cannot be specified unless a first prompt is specified. If prompts are specified, default responses are optional. A 0 value, or an address pointing to a null string (length of 0), in any of the buffer registers indicates no string.

If no error is returned, the specified message is sent to the operator interface subsystem as an initiated operation. To receive responses from the operator, OI\$WAT must be called.

3.10.6.3 Wait for Operator Response (OI\$WAT). The caller uses OI\$WAT to wait for an operator response.

Calling Sequence

```
BLWP @OI$WAT
BYTE Ra,Rb
```

Registers Used

Ra — Address of buffer in which operator response to first prompt is to be placed
Rb — Address of buffer in which operator response to second prompt is to be placed

A zero value in either register indicates no buffer.

Error Codes

- >90FF — SVC error
- >9107 — Address pointer of message is 0
- >9108 — Message buffer is too small
- >9109 — No message outstanding
- >910A — Negative response by operator
- >910B — Previous message timed out without response
- >910E — Operator interface not initialized
- >9110 — Operator interface error returned

Only one operator message may be outstanding at any given moment. No error is given if a task sends a message that completes, and then sends another message without testing for a response to the first message. An operator can cause a negative response to be returned by issuing a KOR (Kill Operator Request) command. If a negative response is returned, OI\$WAT will set the status register to a not equal status. Otherwise, an equal status is returned.

3.10.6.4 End Operator Interface Subsystem Interface Session (OI\$END). Routine OI\$END terminates communication with the operator interface subsystem and releases the LUNO to the operator channel.

Calling Sequence

```
BLWP @OI$END
```

Error Codes

- >90FF — SVC error

After calling this routine, the operator interface may no longer be used until a new call to OI\$BGN is made. Any outstanding operator messages are aborted.

3.10.7 Mailbox Subsystem Interface Routines

The mailbox subsystem interface routines route messages among tasks. The routines are as follows:

- MB\$INT — Initializes mailbox interface
- MB\$SND — Allows a task to send mail to an addressee
- MB\$RCV — Allows a task to receive mail
- MB\$RLS — Allows a task to stop receiving mail

3.10.7.1 Initialize Mailbox Interface (MB\$INT). Routine MB\$INT must be called before any other mailbox interface routines may be called. This routine assigns a LUNO to the mailbox channel and initializes the mailbox interface.

Calling Sequence

```
BLWP @MB$INT
```

Registers Used

- R0 — Error code returned by MB\$INT
- R1 — One of the following:
 - *Address of SVC call block if an error is returned
 - *Points to a site name string on entry

Error Codes

>90FF — SVC error

3.10.7.2 Send Mail (MB\$SND). Routine MB\$SND allows a task to send mail to an addressee.

Calling Sequence

```
BLWP @MB$SND
```

Registers Used

- R1 — Address of buffer that contains message text
- R2 — Address of buffer that contains name of addressee (one to eight characters)

Error Codes

- >90FF — SVC error
- >9100 — No message buffer specified
- >9101 — No addressee buffer specified
- >9102 — Message length of 0 specified
- >9103 — Illegal addressee buffer length
- >9104 — All-blank addressee specified

3.10.7.3 Receive Mail (MB\$RCV). Routine MB\$RCV allows a task to receive mail addressed to any one of up to three names.

Calling Sequence

BLWP @MB\$RCV

Registers Used

- R1 — Address of buffer that contains message text
- R2 — Address of buffer that contains time and date
- R3 — Name list in the format:

< length of list> < name length> < name> ...

Each name can be one to eight characters in length.

The time and date returned is in ASCII string format. It is same format as SCI displays when a Create Message (CM) message is received. It is the time and date when mailbox received the mail, not when the mail was requested through MB\$RCV.

Error Codes

- > 90FF — SVC error
- > 9105 — All blank name specified in name list
- > 9106 — Name length exceeds eight characters
- > 9107 — Name list length exceeds 28 characters
- > 9108 — No name list specified
- > 9109 — Time and date buffer too small
- > 910A — Message buffer too small
- > 910B — No time and date buffer specified
- > 910C — No message buffer specified

3.10.7.4 Release Mailbox (MB\$RLS). Routine MB\$RLS allows a task to stop receiving mail from the mailbox. This routine can receive any mail that has been received since the last call to MB\$RCV.

Calling Sequence

BLWP @MB\$RLS

Registers Used

- R1 — Address of buffer that contains message text
- R2 — Address of buffer that contains time and date

Errors Codes

- > 90FF — SVC error

Writing an SVC Processor

4.1 NEED FOR AN SVC PROCESSOR

About 70 supervisor calls (SVCs) are included with DNOS to perform services and provide access to data structures. However, certain situations require additional, special SVCs. DNOS allows you to write your own SVC processor and include it as part of DNOS during system generation.

4.2 HOW TO WRITE AN SVC PROCESSOR

To add an SVC to DNOS, you must design the call block for the SVC, build several tables, write a processor for the SVC, and include relevant information during system generation.

4.2.1 SVC Call Block

Because of the close relationship between the SVC call block and the SVC processor, only the writer of the processor can design the call block. Except for the first three bytes, you must determine the size and content according to the SVC functions to be performed.

Byte 0 of the call block must contain the SVC opcode. The standard set of SVCs uses opcodes ranging from 0 through > 7F. You can implement SVCs using opcodes from > 80 through > FF. You can specify one or more opcodes using any codes within the user-defined range.

The SVC processor returns a code in byte 1 of the SVC call block. This code is 0 when the SVC completes normally. A code other than 0 is returned when an error occurs or a warning is appropriate.

Byte 2 of the call block contains a subopcode when an SVC supports several operations. When an SVC performs only one operation, you can use byte 2 for any purpose.

To allow for adaptations or extensions to an SVC and its processor, you should include a reserved word at the end of the defined call block. Also, you should make the block an even number of bytes beginning on a word boundary.

4.2.2 SVC Definition Tables

To enable the processor to operate as efficiently as possible, the operating system copies some or all of the call block into a special structure for use by the SVC processor. You must specify how much information to copy and how much to return to the user task in a pair of tables defined for each SVC. These tables are included in a module of definition information for use by system generation. The module can be built in any file but must include the following:

- The IDT name, RPUDAT
- DEF statements for RPUMAX and RPUTAB

- A REF statement for each SVC processor entry point
- A byte named RPUMAX that contains the largest user-defined SVC opcode
- A table named RPUTAB that contains a two-word entry for each SVC opcode in the range > 80 through RPUMAX
- A request description block (RDB) for each user-defined SVC
- A return information block (RIB) for each user-defined SVC that returns data to the calling task

The entries in the table RPUTAB consist of two words each. The first word contains > E000, and the second contains the address of the RDB for the SVC opcode being defined. The first entry in the table is for SVC opcode > 80. Each successive entry is for the next SVC opcode in sequence. If a particular opcode is not defined in the system being generated, the entry in RPUTAB must consist of two words of zero. The RPUDAT module must be assembled, and the object module path-name of the module must be supplied to the system generation program. Figure 4-1 shows an example of RPUTAB that lists two SVCs defined by the user.

An RDB includes the address of the SVC processor, the address of the RIB, and how much information to supply to the SVC processor. Table 4-1 explains the format of an RDB.

Table 4-1. Request Definition Block (RDB) Format

Field Size	Contents
Word	Flags, > 1000 for user-defined SVCs
Word	Address of the SVC processor
Word	Address of the RIB for this SVC (zero if no RIB is defined)
Word	Size of the call block in bytes
Byte	Number of bytes of the call block to be copied by DNOS for the SVC processor (starting at byte 0)
Byte	Zero
Word	Zero

Figure 4-1 shows several RDB definitions for user-defined SVCs.

The operating system uses an RIB to return data from the system copy of the call block to the calling task. If only the error byte of the call block is returned, no RIB is needed. An RIB must be specified in the RPUDAT module when any other information is to be returned. Table 4-2 shows the format of an RIB. The pair of byte fields can be repeated if information is to be returned from several noncontiguous areas in the call block.


```

*-----
* THIS MODULE HAS THE DATA TABLES TO ENABLE PROCESSING OF
* USER-DEFINED SVCS. RPUTAB IS THE TABLE OF RDB AND PROCESSOR
* ADDRESSES FOR THE SVCS. THE SET OF RDB DEFINITIONS FOLLOWS,
* AND RIB DEFINITIONS ARE INCLUDED FOR RELEVANT CASES. IN
* ADDITION, RPUMAX IS DEFINED TO BE THE MAXIMUM USER-DEFINED
* SVC CODE.
*-----
      IDT  'RPUDAT'
      DEF  RPUMAX,RPUTAB          LABELS TO ACCESS USER DATA
      REF  SVC080,SVC082         LABELS OF ENTRY POINTS
RPUTAB DATA >E000              SVC80 - FIND CPU TIME
      DATA RDBU80
      DATA 0                    SKIP SVC81
      DATA 0
      DATA >E000              SVC82 - SPECIAL ADD
      DATA RDBU82
RPUMAX BYTE >82                MAXIMUM USER-DEFINED CODE
*
RDBU80 DATA >1000              FLAGS
      DATA SVC080              PROCESSOR
      DATA RIBU80              RETURN INFORMATION BLOCK
      DATA 6                    MAXIMUM CALL BLOCK SIZE
      BYTE 2                      COPY ONLY TWO BYTES
      BYTE 0                      RESERVED
      DATA 0                      RESERVED
RDBU82 DATA >1000              FLAGS
      DATA SVC082              PROCESSOR
      DATA RIBU82              RETURN INFORMATION BLOCK
      DATA 16                   MAXIMUM CALL BLOCK SIZE
      BYTE 16                     COPY ALL
      BYTE 0                      RESERVED
      DATA 0                      RESERVED
RIBU80 DATA 0                  RESERVED
      BYTE 2,4                    START AT OFFSET 2, RETURN 4 BYTES
      DATA 0                      RESERVED
RIBU82 DATA 0                  RESERVED
      BYTE 2,6                    START AT OFFSET 2, RETURN 6 BYTES
      BYTE 12,4                   AND AT OFFSET 12, RETURN 4 BYTES
      DATA 0
      END

```

Figure 4-1. Format of RPUDAT Module

Table 4-2. RIB Format

Field Size	Contents
Word	Zero
Byte	Offset in the call block from which the return of data should begin
Byte	Number of bytes to return
Word	Zero

Figure 4-1 shows a source module for defining two user SVCs using SVC opcodes > 80 and > 82 with opcode > 81 omitted.

4.2.3 SVC Processor Details

The SVC processor must define its own entry point in a DEF directive. It must save and restore system registers by using two macro calls: SPUSH 1 as the first instruction and SPOP 1 as the last instruction. The processor runs as part of the operating system kernel, making use of an operating system workspace. Upon entry to the processor, the following registers are set:

- Register 1 — Points to the system copy of the requesting call block
- Register 4 — Points to the requester task status block (TSB)
- Register 5 — Points to the requester-saved map file
- Register 10 — Points to an internal operating system stack
- Register 13 — Requesting task workspace pointer
- Register 14 — Requesting task program counter
- Register 15 — Requesting task status register

The SVC processor must not alter registers 10, 13, 14, and 15.

Register 1 contains the address of the system copy of the requester's call block. The processor usually gathers all the information it needs from this copy. The processor alters the copied call block to pass information back to the requesting task. The second byte of the call block should always be used for returning a status code. If necessary, the processor can also access the requester task area to get or return data using long distance instructions, with register 5 as the map file pointer.

When the processor finishes its work, it must return to the operating system by issuing the following:

```
CLR R0
SPOP 1
```

The operating system then returns information as specified in the RIB for the SVC performed. Finally, control passes back to the task that issued the SVC.

Figure 4-2 shows a processor for user-defined SVC >80, corresponding to the definitions in Figure 4-1.

```

TITL 'SVC080 PROCESSOR -- GET EXECUTION TIME'

*****
* THIS EXAMPLE PROCESSOR IS FOR USER-DEFINED SVC >80. IT *
* RETURNS THE AMOUNT OF CPU TIME USED BY THE TASK SO FAR. *
* IT ACCESSES THE FIELD "TSBCPT" IN THE TSB. *
* *
* THE CALL BLOCK HAS THE FORM: *
* *
*      +-----+ *
*      00  !      80      !  ERROR CODE      ! *
*      +-----+ *
*      02  !      TIME EXECUTING SO FAR      ! *
*      +-----+ *
*      04  !      IN INTERNAL CLOCK TICKS    ! *
*      +-----+ *
* *
* UPON ENTRY R1 POINTS TO THE COPY OF THE CALL BLOCK *
* R4 POINTS TO THE TSB OF THE REQUESTER *
*****
IDT 'SVC080'
DEF SVC080          ENTRY POINT
LIBIN DSC.MACROS.TEMPLATE TO USE TSB EQUATES
COPY  DSC.TEMPLATE.ATABLE.TSB ...
LIBIN DSC.MACROS.FUNC   FOR OS FUNCTIONS
SVC080 EVEN
SPUSH 1              SAVE RETURN ADDRESS
MOV @TSBCPT(R4),@2(R1) MOVE EXECUTION TIME
MOV @TSBCPT+2(R4),@4(R1) ...INTO CALL BLOCK
CLR R2              GET A ZERO
MOVB R2,@1(R1)     SHOW NO ERROR
CLR R0              PREPARE FOR RETURN
SPOP 1              RETURN TO DNOS
END

```

Figure 4-2. User-Defined SVC

4.2.4 System Generation Requirements

To include user-defined SVCs, specify the pathname of the RPUDAT object module in response to the following request during system generation:

USER SVC TABLE PATHNAME:

In addition, place the object module for each user-defined SVC processor in a directory called `.$OSLINK.$SGU$.USERSVC` on the data disk used during system generation. This directory is treated as a library when the system is linked; consequently, the processor entry point name must be the same as the file name of the appropriate module in the `.$OSLINK.$SGU$.USERSVC` directory. If a file has several SVC processors in it, each processor entry point name must be listed as either an alias or a file name in the `.$OSLINK.$SGU$.USERSVC` directory.

Writing a DSR

5.1 INTRODUCTION

Although DNOS supports a variety of peripheral devices, some users find it necessary to add a device not supported by the standard software. This section presents a method for writing software to support a nonstandard device.

The software that controls a peripheral device is called a device service routine (DSR). This section supplies information to assist you in writing a DSR for DNOS. The ideas and materials presented are taken directly from the DSRs for the standard devices supported by DNOS. The examples apply to devices connected either through the communications register unit (CRU) or through the TILINE. (TILINE is a registered trademark of Texas Instruments Incorporated.) The CRU is a low-speed, bidirectional, serial data bus. The TILINE is a high-speed, bidirectional, parallel data bus.

This section describes the DNOS I/O subsystem, DSR support routines, and DSR data structures. This section also tells how to write DSRs for asynchronous controllers. These descriptions are not comprehensive; they apply specifically to the problem of writing a DSR to support a nonstandard device. Refer to the *DNOS System Design Document* for further details and diagrams of data structures discussed in this section.

5.2 PREPARATION

To write a DSR, you should be familiar with the following areas:

- Hardware interface for the device
- DNOS I/O subsystem
- Basic data structures
- Computer hardware
- Assembly language

You should also study the process of system generation. Although this information is only indirectly related to writing a DSR, it provides insight into how the operating system interfaces with a DSR.

To help you locate the appropriate material, the following paragraphs describe the I/O subsystem and the basic data structures.

5.3 I/O SUBSYSTEM

The I/O subsystem moves data between any combination of logical and physical I/O resources and programs (tasks) that process the data. A DSR is concerned with the path between the program and the physical I/O resource (that is, the device).

5.3.1 Data Structures

The following templates define the data structures that relate to writing a DSR. To use one of these templates, insert a COPY statement in the source.

In all the data structure pathnames, DSC is a synonym that must be assigned the value of <volume>.\$OSLINK, where volume is the name of the data disk used for system generation. You can examine any of these structures by printing the appropriate file from the DNOS release disk.

Template	Contents
DSC.TEMPLATE.ATABLE.BRO	Buffered request overhead
DSC.TEMPLATE.ATABLE.CDE	Command definition entry
DSC.TEMPLATE.ATABLE.IRB	I/O request block (IRB)
DSC.TEMPLATE.ATABLE.PDT	Peripheral device table (PDT)
DSC.TEMPLATE.ATABLE.KSB	Keyboard status block (KSB)
DSC.TEMPLATE.ATABLE.XTK	Keyboard extension

If you use the KSB and XTK templates in the DSR, you must include the following EQU statements:

KSB BGN	EQU	PDT SIZ	Precedes KSB copy
XTK BGN	EQU	KSBSIZ	Precedes XTK copy

If you are writing a DSR for an asynchronous controller that is to be used with standard DSRs for asynchronous controllers, you must use the following templates:

Template	Contents
DSC.TEMPLATE.ATABLE.DSALLLEX	Asynchronous local extension
DSC.TEMPLATE.ATABLE.DSALLREX	Asynchronous remote extension
DSC.TEMPLATE.ATABLE.AJMPMAC	Subroutine references for an asynchronous hardware service routine (HSR).

The following templates define word and byte constants available to the DSR:

Template	Defined Constants
DSC.TEMPLATE.COMMON.NFER00	BYTE00-BYTE0F
DSC.TEMPLATE.COMMON.NFER10	BYTE10-BYTE1F
DSC.TEMPLATE.COMMON.NFER20	BYTE20-BYTE2F
DSC.TEMPLATE.COMMON.NFER30	BYTE30-BYTE3F
DSC.TEMPLATE.COMMON.NFER40	BYTE40-BYTE4F
DSC.TEMPLATE.COMMON.NFER50	BYTE50-BYTE5F
DSC.TEMPLATE.COMMON.NFER60	BYTE60-BYTE6F
DSC.TEMPLATE.COMMON.NFER70	BYTE70-BYTE7F
DSC.TEMPLATE.COMMON.NFER80	BYTE80-BYTE8F
DSC.TEMPLATE.COMMON.NFER90	BYTE90-BYTE9F
DSC.TEMPLATE.COMMON.NFERA0	BYTEA0-BYTEAF
DSC.TEMPLATE.COMMON.NFERB0	BYTEB0-BYTEBF
DSC.TEMPLATE.COMMON.NFERC0	BYTEC0-BYTECF
DSC.TEMPLATE.COMMON.NFERD0	BYTED0-BYTEDF
DSC.TEMPLATE.COMMON.NFERE0	BYTEE0-BYTEEF
DSC.TEMPLATE.COMMON.NFERF0	BYTEF0-BYTEFF
DSC.TEMPLATE.COMMON.NFWORD	WD0001-WD8000 & MASTAB

The following template contains the pointers to items relevant to the operating system:

DSC.TEMPLATE.COMMON.NFPTR	Pointer segment
---------------------------	-----------------

To assemble templates, you need a set of macros. You can access them by means of a LIBIN statement using the following pathnames:

```
DSC.MACROS.TEMPLATE
DSC.MACROS.FUNC
```

In addition to the system-defined data structures, you can design your own data structure, called the device information block (DIB). To access this structure, use workspace register 4 of the physical device table (PDT). You should set the origin of the DIB to follow the PDT and any system data structures that follow the PDT. The DIB includes any data you wish to access or maintain by using the DSR.

5.3.2 Request Flow

An I/O request enters the I/O subsystem via the supervisor call (SVC) interface. The SVC interface decodes the request and passes it to the I/O subsystem. Routing information is derived from the logical unit number (LUNO) in the I/O request block (IRB). The I/O system checks the access privilege to the physical resource. Routing data and requester identifiers are concatenated to the IRB to form the buffered request block (BRB). The I/O subsystem passes the BRB to the device manager.

The device manager examines the operation code of each request and processes each accordingly. The device manager processes operation codes > 00, > 03, > 05, > 09, > 0A, > 0B, and > 0C. The Open operation codes > 00 and > 03 are checked only for terminals (keyboard devices that use a KSB). The device manager verifies the request buffer and allocates table space in the buffer table area (BTA) for the read operation codes > 05, > 09, and > 0A.

For write operation codes > 0B and > 0C, the device manager copies the data buffer into the BTA after verifying the buffer and allocating space as for read operations. Bits DSFBI and DSFBO in the PDT control allocation of the BTA for each device. The device manager then places the request on a queue associated with the PDT and passes control to the DSR, which processes the request immediately or after completing any prior requests.

While the DSR is processing a request but waiting for an interrupt, control returns to the operating system. The system executes programs during this time until the DSR receives an interrupt. An interrupt returns control to the DSR for further processing.

When the DSR terminates the current request, the next request (if any) is passed to the DSR. The completed request is placed in a queue related to the program. The next time the program executes, the completed request is returned to the program. This completes the cycle for an I/O request.

5.3.3 Device Interrupt Decoder

After the DSR initiates the device operation associated with the request, control returns to the system until an interrupt from the device causes the DSR to resume processing the request. Therefore, the programmer who is writing the DSR should know how the system uses the system interrupt decoder to process an interrupt.

The interrupt decoder loads the DSR map file and transfers control to the DSR for the interrupting device. The methods of handling interrupt signals are as follows:

- Single device per interrupt level
- Multiple devices per interrupt level
- Multiple devices in an expansion chassis
- Single device or multiple devices on a multiplexed device controller

Subsequent paragraphs describe typical system-interrupt decoder routines. The system generation process builds these routines, and users cannot modify them.

5.3.3.1 Single-Device Interrupt. Figure 5-1 shows an example of the interrupt decoder for a single device at interrupt level 13. The workspace address (WSPD) and the interrupt decoder execution address (PCS) are stored at locations > 34 and > 36, respectively. The first six registers (R0 through R5) of the workspace are not used. Register R6, which is the service flag, is set to 1, and register R7 contains 0. Register R8 contains WPdsr, the address of the workspace for the DSR. Register R9 contains SG3BGN, the execution address of the interrupt service routine (ISR) within the DSR. Register R10 contains MPdsr, the address of the map file for the DSR.


```

        DATA WSPD,PCS                LOCATION >34 AND >36
        .
        .
        .
WSPD    EQU    $-12                    R0-R5
        DATA 1,0,WPdsr,SG3BGN,MPdsr  INPRPT ENTRY
        DATA 0,0,0,0,0                R11-R15
        .
        .
        .
PCS     MOV    @CURMAP,R11              SAVE CURRENT MAP FILE
        MOV    R10,@CURMAP              SET UP DSR MAP FILE
        LMF   R10,0
        BLWP  R8                        ENTER DSR
        JMP   RETURN

```

Figure 5-1. Interrupt Decoder for Single Device

When an interrupt occurs, the current status is saved in registers R13, R14, and R15 of WSPD. The interrupt decoder begins execution by saving the current map file address in register R11. The routine places the address of the DSR map file in location CURMAP. The DSR map file is loaded and control is transferred to the ISR. On return from the ISR, the interrupt decoder branches to a return routine at location RETURN.

5.3.3.2 Multiple-Device Interrupt. Figure 5-2 shows an example of the interrupt decoder for multiple devices at interrupt level 10. Locations >28 and >2A contain the address of the workspace (WSPA) and the interrupt decoder execution address (PCM), respectively. The first six registers (R0 through R5) of the workspace are not used. Register R6, which is set to 0 initially, is the service flag. It is set to 1 at the completion of the servicing of the interrupt. Register R9 contains the address of a table (TABLA) that contains pairs of words. The table has extra pairs for additional devices and is terminated by a word that contains 0 (DATA 0). The first word of each pair contains the CRU address of the bit that tells you which device caused the interrupt. The second word of the pair is the address of a three-word data structure (IVxx01 or IVxx02) that contains the workspace address (WPdsr), the execution address (SG3BGN), and the map file address (MPdsr) for the ISR.

```

        DATA WSPA,PCM          LOCATION >28 AND >2A
        .
        .
        .
WSPA    EQU    $-12             R0 -R5
        DATA 0,0,0,TABLA,0,0,0,0,0,0    R6-R15
        .
        .
        .
TABLA   EQU    $              MULTIPLE INTERRUPT DECODER TABLE
        DATA <cru interrupt bit address>
        DATA IVxx01
        DATA <cru interrupt bit address>
        DATA IVxx02
        DATA 0                LOGICAL END OF TABLE
        .
        .
        .
IVxx01 DATA KBdsr,SG3BGN,MPdsr
IVxx02 DATA KBdsr,SG3BGN,MPdsr
        .
        .
        .
PCM     MOV    @CURMAP,R11      SAVE CURRENT MAP FILE POINTER
        MOV    R9,R8           GET THE TABLE ADDRESS
        CLR    R6              SET SERVICE FLAG TO ZERO
PCM10   MOV    *R8+,R12        IS THE TABLE EXHAUSTED?
        JEQ   RETURN          AND EXIT
        TB    0               DID THIS GUY DO IT?
        JNE   PCM20          NOT I, SOMEONE ELSE DID IT
        MOV    *R8,R7          GET ENTRY VECTOR
        MOV    @4(R7),R10      PICK UP THE NEW MAP ADDRESS
        MOV    R10,@CURMAP     CHANGE MAPS
        LMF   R10,0
        BLWP  *R7              ENTER THE DSR
PCM20   SETO  R6              INDICATE AN INTERRUPT SERVICED
        INCT  R8              NEXT TABLE ENTRY
        JMP   PCM10

```

Figure 5-2. Interrupt Decoder for Multiple Devices

When an interrupt occurs, the current status is saved in registers R13, R14, and R15 of WSPA. The interrupt decoder begins execution by saving the current map file pointer in register R11. The decoder then moves the address of table TABLA into register R8 and clears the service flag in register R6. The device interrupt bit must be tested to determine if the device interrupted. The CRU address of this device is moved into register R12. If the address is 0, the interrupt decoder branches to the return code. Otherwise, the device interrupt bit is checked. If it is 0, the next device interrupt bit is checked. A nonzero value identifies the interrupting device, and the address in the next word (IVxx01 or IVxx02) is moved into register R7. The DSR map file address is moved into register R10. The routine then moves the DSR map file address into location CURMAP, loads the map file, and branches to the ISR. When the ISR returns control to the interrupt decoder, the interrupt decoder checks the remaining devices in TABLA for interrupts.

5.3.3.3 Expansion Chassis Interrupt. Figure 5-3 shows an example of the interrupt decoder for devices in the expansion chassis on the first expansion card at interrupt level 7. Locations >1C and >1E contain the address of the workspace (WSP7) and the interrupt decoder execution address (PCE), respectively. The first six registers (R0 through R5) of the workspace are not used. Registers R6 and R12 contain the CRU base address of the expansion card.

Location EXPTST contains the first of a table of mask words that correspond to flag bit positions for the expansion chassis. Table ETAB contains the address (CTAB1) of a table for the first expansion chassis on the first card. Table CTAB1 contains addresses of three-word data structures (IVxx03 or IVxx02) that contain the workspace address (WPdsr), the execution address (SG3BGN), and the map file address (MPdsr) for the ISRs. The table also contains the address of a table (IVX110) for multiple devices on the same interrupt position and a flag word for each address in the table. The flag for each address follows the address. It is set to 0 for three-word data structure addresses and to -1 for the multiple device table address. The multiple device table contains CRU addresses for the interrupt bits and addresses of three-word data structures (IVxx04 and IVxx05) for ISRs for those devices.

```

          DATA WSP7,PCE          LOCATION >1C AND >1E
          .
          .
          .
WSP7     EQU    $-12              R0 -R5
          DATA >1F00,0,0,0,0,0,>1F00,0,0,0          R6 -R15
          .
          .
          .
EXPTST   DATA >4000,>2000,>1000,>800    TEST BIT FOR EXP CHASSIS
          .
          .
          .
ETAB     EQU    $                EXPANSION CHASSIS TABLE
          DATA CTAB1,0,0,0,0,0,0,0,0
          .
          .
          .

```

Figure 5-3. Expansion Chassis Interrupt Decoder (Sheet 1 of 3)

PCE	EQU	\$	EXPANSION CHASSIS ENTRY
	MOV	@CURMAP,R11	SAVE CURRENT MAP FILE POINTER
PCE10	STCR	R9,0	FIND CHASSIS THAT INTERRUPTED
	MOV	R9,R10	
	SRL	R10,6	
	ANDI	R10,6	
	COC	@EXPTST(R10),R9	IS THAT CHASSIS PRESENT?
	JEQ	IX	NO, CRASH!!!
	A	R8,R10	ADD CARD LEVEL
	MOV	@ETAB(R10),R10	IS THE CHASSIS DEFINED?
	JEQ	IX	NO, TAKE IT DOWN
	ANDI	R9,>7C	INTERRUPT POSITION
	A	R9,R10	INDEX INTO TABLE
	MOV	*R10+,R9	INTERRUPT POINTER
	JEQ	IX	NOTHING THERE
	MOV	*R10,R7	CHECK FLAG
	JNE	PCE20	MULTIPLE DEVICES THERE
	MOV	@4(R9),R10	DSR MAP BECOMES CURRENT MAP
	MOV	R10,@CURMAP	
	LMF	R10,0	
	BLWP	*R9	ENTER THE DSR
PCE15	MOV	R6,R12	RESTORE THE CARD BASE
	TB	15	ANY MORE INTERRUPTS?
	JEQ	PCE10	YES, GET THEM NOW
	JMP	RETURN	OTHERWISE, EXIT
PCE20	MOV	*R9+,R12	END OF TABLE?
	JEQ	PCE15	YES, BACK TO THE CARD LEVEL
	MOV	*R9+,R7	GET THE ENTRY VECTOR
	TB	0	IS THIS IT?
	JNE	PCE20	NO, KEEP LOOKING
	MOV	@4(R7),R10	PICK UP MAP POINTER
	MOV	R10,@CURMAP	CHANGE MAPS
	LMF	R10,0	
	BLWP	*R7	ENTER THE DSR
	JMP	PCE20	

Figure 5-3. Expansion Chassis Interrupt Decoder (Sheet 3 of 3)

When an interrupt occurs, the current status is saved in registers R13, R14, and R15 of WSP7. The interrupt decoder begins execution by saving the current map file address in register R11. The interrupt decoder reads the expansion coupler status word into register R9. The decoder calculates the ID of the interrupting chassis (times 2) in register R10. Using a mask from table EXPTST, the decoder verifies that the expansion chassis is connected. If it is not, the interrupt decoder branches to the illegal interrupt routine.

Register R10 is then used to index into the table ETAB to get the address of the chassis table. If the pointer is 0, the interrupt decoder branches to the illegal interrupt routine. If not, the device interrupt position is added to the chassis table address to obtain the index of the address corresponding to the interrupt position in the chassis table. If the address is 0, the interrupt decoder branches to the illegal interrupt routine. If not, the decoder tests the flag associated with the address.

If this flag is a nonzero value, the decoder branches to location PCE20 to process multiple devices. If the flag is 0, the decoder moves the DSR map file pointer into register R10 and into the current map file pointer, CURMAP. The decoder loads the DSR map file and transfers control to the ISR. When control returns from the ISR, the decoder restores the proper CRU address in register R12 and tests the expansion chassis interrupt bit for another interrupt. When another interrupt exists, the decoder branches to location PCE10 to decode the outstanding interrupt. Otherwise, the decoder branches to the return code.

The multiple-device routine at location PCE20 is similar to the routine described in the preceding paragraph. It loads the CRU address and tests for 0 at the end of the table. Then, the routine loads the data structure address corresponding to the CRU address. The routine tests the interrupt bit and returns to location PCE20 when the bit is 0. When the interrupting device is located, the routine loads the map file and transfers control to the ISR as previously described. On return from the ISR, the routine branches to location PCE20 to process any additional interrupt at the interrupt position.

5.3.3.4 Asynchronous Multiplexer Interrupt Decoder. Use this decoder for CI403 and CI404 multiplexers in the following cases:

- They use a single interrupt.
- They share interrupts in the main chassis.
- They share interrupts in the expansion chassis.

Figure 5-4 shows an example of the interrupt decoder for devices on a multiplexer(s) that has an interrupt level of 11 in the main chassis. Location >2C contains the address of the workspace (WSPB) and location >2E contains the decoder entry point (PCA). The decoder uses the first three registers (R0 through R2 of WSPB) to pass controller information to the asynchronous DSR. In R10, there is a pointer to the controller table(s) (ACTLB) which the system generation builds for each controller that shares an interrupt level of 11. The decoder places the TILINE address of the controller in R12 and uses the remaining registers as temporary registers.

Location ACTLB contains three word (word 0 through word 2) of controller information for each controller that shares interrupt level 11. The table is terminated with a zero. Word 0 contains the TILINE address of the controller. Word 1 is a pointer to the channel table (MUXB01). Word 2 is the result of taking the maximum channel number of the controller and multiplying that number by eight.

Location MUXB01 contains four words (word 0 through word 3) of information for each channel of the controller (for a channel that was not specified during system generation, the four words associated with the channel each contain zero). Word 0 of the channel table contains a pointer to the DSR interrupt workspace (KBdsr). Word 1 contains the DSR address (SG3BGN). Word 2 contains a pointer to the DSR map file (MPdsr). Word 3 contains the channel number.

```

          DATA WSPB,PCA          LOCATION >2C,>2E
          .
          .
WSPB     BSS     20                R0  R9
          DATA  ACTLB-4          R10 -- CONTROLLER TABLE ADDRESS
          BSS     10                R11 -- R15
          .
          .
ACTLB    EQU     $                MULTIPLE INTERFACE TABLE
          DATA  >F980            MUX INTERRUPT WORD ADDRESS
          DATA  MUXB01           MUX CHANNEL TABLE ADDRESS
          DATA  3*8              MAX MUX CHANNEL ID * 8
          DATA  0                LOGICAL END OF TABLE
MUXB01   EQU     $                CHANNEL TABLE
          DATA  KBdsr,SG3BGN,MPdsr,0
          DATA  KBdsr,SG3BGN,MPdsr,1
          DATA  0,0,0,0          DUMMY CHANNEL ENTRY
          DATA  0,0,0,0          DUMMY CHANNEL ENTRY
          .
          .
SLW3     EQU     R3*2             SLAVE WORD THREE ADDRESS
PCA      MOV     @CURMAP,R11      SAVE CURRENT MAP FILE
          MOV     R10,R9          GET CONTROLLER TABLE ADDRESS
PCA015   AI     R9,4             GET CONTROLLER TABLE ENTRY
PCA020   MOV     *R9+,R12
          JEQ    PCARTN           IF TABLE EXHAUSTED, EXIT
          JGT    PCA015           IF CONTROLLER NOT PRESENT, NEXT CONTROLLER
          SETO   R4              SET LOAD MAP FILE FLAG
          MOV    *R12,R8          DID THIS CONTROLLER DO IT?
          JGT    PCA015           NO -- NEXT CONTROLLER
          MOV    *R9+,R7          GET STATION LIST ADDRESS
          MOV    *R9+,R6          GET CHANNEL COUNT (COUNT*8)
PCA025   MOV    @SLW3(R12),R0    IS THE INTERRUPT INVALID?
          JLT    PCA020           YES -- NEXT CONTROLLER
          MOV    R0,R1            SAVE DATA WORD
          MOV    R0,R2            SAVE DATA WORD
          MOV    R4,R4            RELOAD MAP FILE?
          JLT    PCA027           YES -- DON'T TEST FOR SAME CHANNEL
          CZC   @WD0800,R0       SAME CHANNEL?
          JEQ   PCA027           YES
          SETO   R4              FORCE MAP FILE LOAD

```

Figure 5-4. Asynchronous Multiplexer Interrupt Decoder (Sheet 1 of 2)

PCA027	SLA R0,8	MOVE DATA TO LEFT BYTE
	SRL R1,12	STATUS CODE TO BITS 13,14,15
	CI R1,>0004	TIMER OR TRANSPARENT FIFO STATUS?
	JEQ PCA050	YES -- PROCESS
	SRL R2,5	8 * CHANNEL NUMBER
	ANDI R2,7*8	ISOLATE CHANNEL NUMBER * 8
	C R2,R6	ADDRESS OUT OF RANGE?
	JH PCA100	YES -- ILLEGAL CHANNEL NUMBER
	MOV R7,R8	SAVE CHANNEL TABLE ADDRESS
	A R2,R8	GET CHANNEL ENTRY VECTOR ADDRESS
	MOV @4(R8),R5	GET MAP FILE ADDRESS
	JEQ PCA025	IF CHANNEL NOT GENNED, NEXT CHARACTER
	ABS R4	CHANGE MAP FILE?
	JGT PCA028	NO
	MOV R5,@CURMAP	YES -- SET MAP FILE POINTER
	LMF R5,0	LOAD DSR MAP FILE
PCA028	BLWP *R8	ENTER DSR
	JMP PCA025	TRY THIS CONTROLLER AGAIN
PCA050	CLR R2	CLEAR CHANNEL NUMBER
	JMP PCA060	BYPASS CHANNEL INCREMENT
PCA055	AI R2,1*8	GET NEXT CHANNEL ENTRY
	C R2,R6	ADDRESS OUT OF RANGE?
	JH PCA110	YES -- CHECK CONTROLLER AGAIN
	LI R1,>0004	RELOAD STATUS CODE
PCA060	MOV R7,R8	SAVE CHANNEL TABLE ADDRESS
	A R2,R8	GET CHANNEL ENTRY VECTOR ADDRESS
	MOV @4(R8),R5	GET MAP FILE ADDRESS
	JEQ PCA055	IF CHANNEL NOT GENNED, NEXT CHANNEL
	MOV R5,@CURMAP	SET MAP FILE POINTER
	LMF R5,0	LOAD DSR MAP FILE
	BLWP *R8	ENTER DSR
	JMP PCA055	NEXT CHANNEL
1CA100	EQU \$	ILLEGAL CHANNEL NUMBER
	INC R3	INCREMENT NUMBER OF ILLEGAL INTERRUPTS
PCA110	SETO R4	SET CHANGE MAP FILE FLAG
	JMP PCA025	NEXT CONTROLLER

Figure 5-4 Asynchronous Multiplexer Interrupt Decoder (Sheet 2 of 2)

When an interrupt occurs, the present status is saved in R13, R14, and R15 of WSPB. When the interrupt decoder begins execution, it saves the current map file address by placing it in R11. The interrupt decoder searches for the controller which generated the interrupt by placing Word 0 of the controller information table (ACTLB) into R12 and then reading slave word 0 of the TPCS. The controller that has an interrupt pending will have a negative value in slave word 0. Once the decoder finds the controller with the interrupt pending, it looks at word 1 of the appropriate controller table to find the pointer to the channel tables (MUXB01). This pointer indicates where the DSRs for each channel are located. The decoder then determines which channel caused the interrupt and picks up the proper channel DSR mapfile (MPdsr) and loads it. The decoder enters the DSR by executing a Branch and Link Workspace Pointer (BLWP) instruction, using the DSR workspace (KBdsr) and program address (SG3BGN). When the DSR returns control to the interrupt decoder, the interrupt decoder checks the remaining controllers at ACTLB for pending interrupts.

5.3.3.5 Return Routine. The return routine in Figure 5-5 tests the service flag and branches to the illegal interrupt routine when the flag is 0, indicating that no interrupt has been serviced. Otherwise, the routine moves the saved map file address to location CURMAP and loads the map file. It transfers to a routine at location NFTRTN that checks processing conditions and executes a Return to Workspace Pointer (RTWP) instruction. This restores the machine status to its pre-interrupt status.

```

RETURN  MOV   R6,R6           TEST SERVICE FLAG
        JEQ   IX             NOTHING SERVICED IS AN ERROR
PCARTN  MOV   R11,@CURMAP     RESTORE OLD MAP FILE POINTER
        LMF   R11,0          RESTORE OLD MAP
        B     @NFTRTN        RETURN TO OS

```

Figure 5-5. Interrupt Decoder Return Routine

5.3.3.6 Illegal Interrupt Routine. The illegal interrupt routine in Figure 5-6 stores the status register contents in register R11 and calculates the level of the illegal interrupt. The routine then moves the level into location LEVEL to form the crash code; finally, the routine branches to the system crash routine using the transfer vector at location NFCSRSH.

```

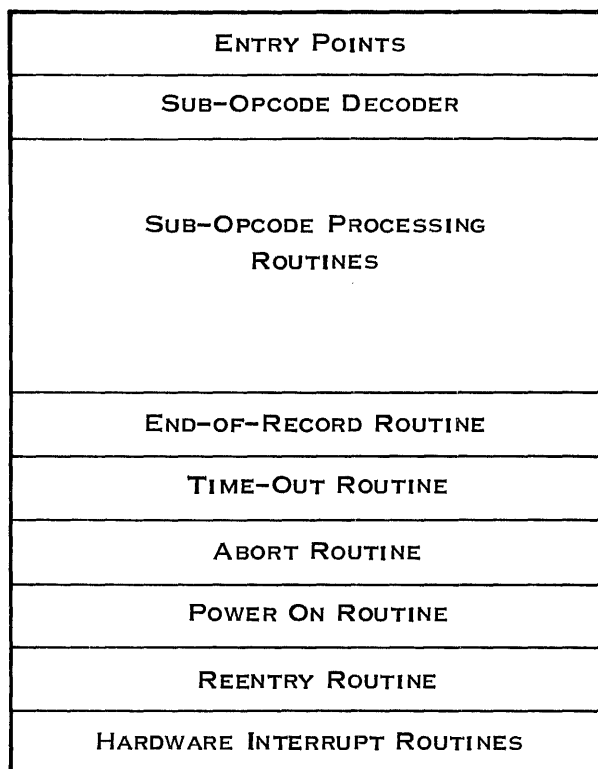
IX      STST  R11
        ANDI  R11,>F
        INC  R11           R11 = BAD INTERRUPT LEVEL
        SOC  R11,@LEVEL
        BLWP @NFCSRSH     BOGUS INTERRUPT → CRASH
LEVEL   DATA >10        CRASH CODES >13 → 1F

```

Figure 5-6. Illegal Interrupt Routine

5.4 DEVICE SERVICE ROUTINES

The following paragraphs describe DSRs, including their design criteria. Figure 5-7 shows the overall organization of a DSR that interfaces directly between a device and DNOS. A DSR that interfaces between an asynchronous controller and DNOS can be structured according to Figure 5-8. Any unique properties of this type of DSR are described in a later description of DSRs for asynchronous devices.



2279421

Figure 5-7. DSR Structure

The entry points to the DSR are described first, along with the routines to which they branch the body of the DSR (the subopcode decoder, the subopcode processing routines, and the end-of-record routine) is described next. The set of DSR support routines that the DSR can call when required is described last.

5.4.1 Design Criteria

To service multiple devices of the same type on the same system, you must design the code of the DSR to be shared and reentrant. The DSR must service each interrupt very quickly. It must appear to service multiple device interrupts simultaneously. However, only one interrupt at an interrupt level can be serviced at one time. Subsequent interrupts must wait for servicing until the current one has completed.

Each PDT maintains information about a single physical device for the I/O subsystem, and each physical device requires a PDT. Part of the PDT is the DSR workspace. Between the processing of requests for the device and interrupts from the device, data is stored either in the PDT workspace registers or in extensions to the PDT. Use registers R1 through R4 and R12 through R15 of the PDT only for their intended purposes, as described in the PDT template. The following registers are available for exclusive DSR use: R0 and R5 through R11.

The DIB is the extension storage area of the PDT. Any data stored in the DSR can be destroyed as the DSR services other devices. Even if your present configuration has only one device for which you are writing the DSR, you should provide for possible expansion to several of these devices. The DSR should not modify any of the DSR code.

The keyboard status block (KSB) maintains device information for keyboard devices. If you answer YES to the KEYBOARD? prompt during system generation of special devices, your device is generated as a keyboard device. In this case, you must include a KSB as the first part of the DIB. The KSBSN field in your KSB is initialized to your station number by system generation. (All devices with a keyboard are given device names that have the form STxx.)

The KSB contains the DSR interrupt workspace for keyboard devices. Like the PDT, the KSB can be extended. The XTK is an example of a KSB extension.

Figure 5-8 shows the logical structure of the DSR in relationship to the operating system. Model 990 Computer hardware maps memory in three segments defined in a map file:

- Interrupt trap addresses and system root (first segment)
- Data buffer (second segment)
- DSR (third segment)

The upper limit of the first segment is shown as XXXX because the size of this segment varies. This value is defined during system generation and can be found in JCASTR in the NFPTR template. The data buffer is in the BTA for CRU-type devices (swapping permitted); it is in the requesting task for TILINE devices (swapping not permitted). The map file that defines the segments for the DSR applies only while the DSR is executing. The system performs mapping. You should be concerned with mapping only if the DSR must map buffers in or out.

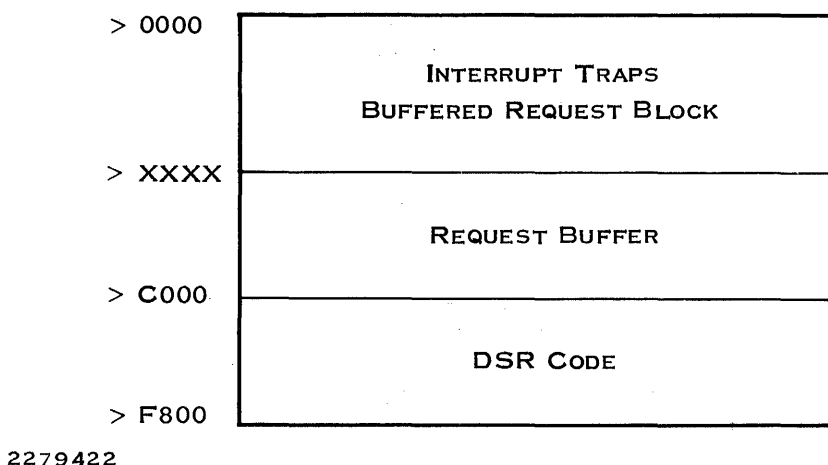


Figure 5-8. Memory Map for DSR Execution

5.4.2 DSR Entry Points

The DNOS operating system can enter the DSR through one of the following entry points:

- Hardware interrupt — Branches to the routine that processes interrupts from the device
- System interrupt — Branches to the routine that processes reentry requests
- Power up — Branches to the routine that initializes the device
- Request abort — Branches to the routine that forces error termination of the current I/O request
- Request time-out — Branches to the routine that processes a device time-out
- I/O Request Processor — Branches to the routine that processes priority requests
- Priority Schedule Interrupt — Branches to routine that processes priority reentry requests.

The I/O subsystem requires that branch instructions for these entry points be placed at the beginning of the DSR (relative address 0) in a specific sequence. The first code of the DSR must contain the branch instructions for the entry points, as given in Table 5-1.

Table 5-1. DSR/TSR Entry Points

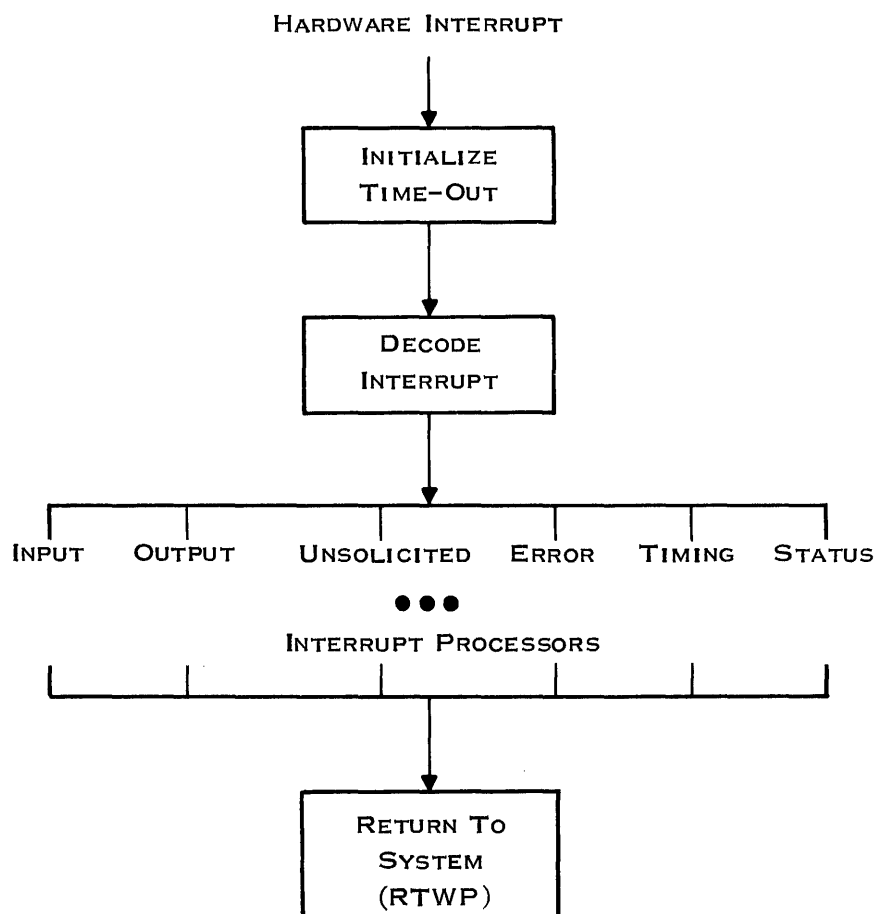
Address	Code	Meaning
> 0000	B @HINT	Hardware interrupt
> 0004	B @SINT	System interrupt (delayed reenter me)
> 0008	B @PWRUP	Power up initialization
> 000C	B @ABORT	I/O abort
> 0010	B @TIMOUT	Time-out

Since these entry points are required to be at absolute locations, no data or subroutine code can precede these instructions.

When a hardware interrupt occurs, the interrupt decoder executes a Branch and Link Workspace Pointer (BLWP) instruction in order to pass control to the DSR. The interrupt mask is set to the interrupt level of the device minus one to prevent another device interrupt. If a device interrupt enters the DSR, it destroys the current context saved in the DSR registers (R13, R14, and R15).

5.4.2.1 Hardware Interrupt Entry Point. The first branch instruction transfers to the routine that processes the hardware interrupts. This routine is the ISR. Figure 5-9 shows the functions of the ISR, which are to initialize the time-out counter, decode the interrupt, reset and process the interrupt, and return to the interrupted program.

The time-out counter is initialized by moving the value in the field PDTTM1 to the field PDTTM2. You can select the time-out option during system generation. The counter should be initialized in any case.



2279423

Figure 5-9. Hardware Interrupt Processing

Each type of device can have several types of interrupts. Each type of interrupt is processed differently. Figure 5-9 shows six types, although all types do not apply to every device. The types are as follows:

- Input
- Output
- Unsolicited
- Error
- Timing
- Status

An input interrupt occurs when the device has data to be transferred to the system. This type of interrupt occurs after a read operation has been requested and prior to completion of the operation.

An output interrupt occurs when a device is ready to receive data from the system. This type of operation occurs after a Write operation has been requested and prior to completion of the operation.

An unsolicited interrupt occurs when the device requires service (for example, has data to be transferred to the system) other than during the servicing of an I/O request.

An error interrupt occurs when the device has error data to be transferred to the system. This type of interrupt can occur during any operation.

A timing interrupt occurs when the device has a timing signal required by the system.

A status interrupt occurs when the device has status information to transfer to the system.

The ISR must include an interrupt decoder to identify each type of interrupt that applies to the device and to transfer control to the interrupt processor for the proper type.

For keyboard devices that use the KSB as the workspace for the input ISR, three supportive routines are available to the DSR: IOFCDT, PUTEBF, and PUTCBF. Subsequent paragraphs describe these routines. The following example outlines a typical input interrupt processor:

- Move a character from the device into R10.
- Check for input errors and process accordingly.
- Reset the input interrupt.

```

*-----
*   Check for bidding a task from a DSR
*-----
LAB010  BL    @IOFCDT
        BYTE >1B,>0D
        BYTE >00,>00
        DATA <special-processing-done address>

Process the character

*-----
*   If the character is an event character
*-----
        BL    @PUTEBF
        DATA <buffer-full address>
        JMP   <clean-up address>

*-----
*   Else buffer the data character
*-----
        BL    @PUTCBF
        DATA <buffer-full address>

Do necessary clean up.

RTWP

```

5.4.2.2 Delayed Reentry Point. The second branch instruction transfers control to the routine that services requested system interrupts. It is called the RE-ENTER-ME routine. The branch instruction transfers control to this routine when the system receives an interrupt approximately 50 milliseconds after the DSR sets bit DSFREN in the PDT. The I/O subsystem resets the bit on entry to the DSR. The DSR writer can use this feature as a timer for long delays during I/O or as a signaling method between the ISR and the DSR.

5.4.2.3 Power-Up Initialization Entry Point. The third branch instruction transfers to the routine that initializes the device upon power-up or power restoration. This routine must initialize the device interface to the proper state.

5.4.2.4 Abort Entry Point. The fourth branch instruction transfers to the routine that aborts a request. When a task is aborted, all I/O requests on all LUNOs must be aborted. The I/O subsystem attempts to find all of the requests and places a code (> 10) in the error byte of each request. The I/O subsystem then notifies the DSR of an aborted request by entering the DSR at the abort entry. This routine must clean up the processing of the request and return it to the I/O subsystem.

5.4.2.5 Time-Out Entry Point. The fifth branch instruction transfers to the time-out routine. The associated interrupt occurs when the device does not respond within a time limit. This function is enabled by specifying a time-out period when the system is generated. The I/O subsystem initializes the time-out count on initial request entry, and the hardware interrupt entry must reset it on each entry. The time-out routine must terminate the request.

5.4.2.6 Initial Request Entry Point. The sixth branch instruction is the request entry point. The DSR is entered at this point when the I/O subsystem has a request for the device and the PDT is not busy. Before entering the DSR, the I/O subsystem initializes the following locations:

- Register R1 (location PDTPRB) and location PDTSRB contain the address of the SVC opcode byte in the BRB.
- The data buffer address field of the BRB contains the address of the mapped in data buffer (IRBDBA) template.
- Location PDTTM2 contains the value in location PDTTM1 and indicates that the time-out count has been initialized. This is done by the I/O subsystem before entering the DSR.

5.4.2.7 Priority Scheduler Entry Point. The seventh branch instruction transfers to the DSR priority scheduler. This mechanism reenters the DSR after all interrupt processing to the system is complete but before the task scheduler initiates any task execution. This is the most direct reentry path to the DSR. Its use is intended only for high-priority interrupt processing. If you use the mechanism arbitrarily, you might interfere with other devices that use this fast reenter me entry point.

5.4.3 Body of the DSR

The body of the DSR consists of the subopcode decoder, the subopcode processing routines, and the end-of-record routine (see Figure 5-7).

The subopcode decoder normally calls DSR support routine BRSTAT to decode the subopcode. Routine BRSTAT also collects information for online diagnostics.

The subopcode processing routines process the I/O requests by translating them into device operations. The number of routines required depends on the device, but usually processing routines for open, write, read, and close operations are provided. The device characteristics determine what each routine does. The programmer writing a DSR for a device must design these routines.

After the subopcode processing routines have performed the appropriate operations, the DSR must call routine ENDRCD to return the request to the calling task. On return from ENDRCD, the DSR performs any necessary clean up and executes an RTWP instruction to return to the I/O subsystem.

5.4.4 Bidding a Task From the DSR

DSR support routine IOFCDT provides the capability of bidding a task by a DSR for keyboard devices that use the KSB as the ISR workspace. The routine bids a specified task when you enter a predefined sequence of characters. The first character in this sequence is referred to as the arming character.

A DSR for a keyboard device that does not use the KSB as the workspace or for a device that does not have a keyboard can also bid a task in response to a defined input sequence. The DSR must perform the functions of IOFCDT or an appropriate subset of those functions.

The task to be bid from a DSR is defined in an entry in a command definition table (CDT). The command definition entry (CDE) contains the LUNO assigned to the program file that contains the task to be bid and the ID of the task. It also contains the ASCII code of the character that is entered to bid the task if more than one entry defines more than one task to be bid.

Routine IOFCDT checks for an arming character before checking the CDT of the device and bidding the task when the corresponding character has been entered.

For devices without a keyboard, the DSR must check for an arming character, if one is required. It must also select the correct bid character. Bidding the task consists of placing the PDT of the device on a queue. First, the DSR must test the value in location PDTCHR of the PDT. When the value is not equal to 0, a task bid is pending and another is not allowed.

Whether to delay bidding the task until the previously requested bid has been completed or to ignore the bid request is a decision that depends on the nature of the DSR and the task being bid. When the value in location PDTCHR is 0, put the bid character in PDTCHR and place the PDT on the queue beginning at location BIDREQ in common segment NFPTR. Set the interrupt mask to level 2 and locate the end of the queue. Location BIDREQ contains the address of location PDTBQ in the first PDT on the queue. That address contains 0 or the address of location PDTBQ in the next PDT. Locate the end of the queue by testing for 0 in location PDTBQ of each PDT. When you find the end, replace the 0 with the address of location PDTBQ of the PDT being serviced and set location PDTBQ in that PDT to 0. Then, restore the interrupt mask to the proper value for the device being serviced.

A system has 25 CDTs. The Set Device Parameters suboperation in the I/O SVC defines the CDT/CDE set for a device. All video display terminal (VDT) devices have the exclamation mark (!) character defined to bid SCI and the CONTROL X sequence defined to abort a task associated with the terminal. You can change these or add more by using the Add CDE to CDT suboperation in the I/O SVC or by using the Modify Command Definition Table (MCDT) command.

5.4.5 Multiplexing Hidden Request Queue

For a DSR that must multiplex its input and output, a queue anchor (PDTHQR) is included in the PDT. When a DSR needs to receive a second request, it must appear to the I/O subsystem to be not busy. The DSR achieves this by mapping out the current request and clearing PDTSRB. It must then keep the first request available by queuing it to the first hidden request queue, PDTHRQ, using the link word BROBRO in the BRB. (See template .ATABLE.BRO to find link word BROBRO.)

In this way, the I/O subsystem can find a request being aborted and flag the error byte with a hexadecimal 10. During an abort, the DSR is entered at the abort entry and must examine PDTHRQ and abort the requests marked with an error code of >10. When a request is removed from the hidden request queue, PDTSRB must be set to the request address to indicate that the device is busy.

If you design a DSR to multiplex two or more requests at the same time, you must be careful about its accessing a buffer. Only the buffer for the request given to the DSR is mapped into the DSR address space. The mapping information for the other request remains with the request. Therefore, the subroutine IOMPOT must be called to map out a request buffer before inserting the request on the queue anchor PDTHRQ. R1 must point to the SVC and error code byte of the BRB. To map the request buffer into the DSR address space, the subroutine IOMPIN must be called. R1 must point to the word of the BRB that contains the SVC code and error byte. Neither of these subroutines modifies PDTSRB.

5.5 DSR SUPPORT ROUTINES

DNOS provides a set of support routines that a DSR can call to perform certain functions. In the following descriptions of these routines, the module pathnames begin with the synonym VOL. Assign the value <volume>.\$S\$OSLINK (in which volume is the volume name of the system data disk) to synonym VOL prior to accessing these modules.

5.5.1 Branch Table Processor Routine

The DSR calls the branch table processor (BRSTAT) to decode the subopcode in the BRB and transfer control to the subopcode processing routine in the DSR. BRSTAT also collects information for online diagnostics.

Defined in Module

VOL.IOMGR.OBJECT.IONRCD

Entry

WS — PDT
R1 — Pointer to the BRB at the SVC opcode byte
R4 — Pointer to the PDT at PDTPDT
R10 — Pointer to statistics table

Calling Sequence

BL @BRSTAT
DATA max # of processors
DATA error return address
DATA processor address for subopcode 0
DATA processor address for subopcode 1
.
.
.
DATA processor address for subopcode n

Statistics Table

BYTE offset into PDT for subopcode 0
 BYTE offset into PDT for subopcode 1
 BYTE offset into PDT for subopcode 2
 .
 .
 .
 BYTE offset into PDT for subopcode n

Exit

R0 — Modified
 R10 — Modified

The online diagnostics require counts of physical I/O operations. These counts are maintained in the following PDT offsets:

PDTRC	Offset to Read operation count
PDTWC	Offset to Write operation count
PDTMC	Offset to miscellaneous operation count

Routine BRSTAT uses a statistics table to associate the appropriate offset with each subopcode. Build the table in the DSR using the offsets in the preceding list. Use the miscellaneous operation offset (PDTMC) for a subopcode that performs a physical device operation other than a Read or Write. Enter 0 in the table for any subopcode that does not perform a physical device operation.

The statistics table contains an entry (either an offset or 0) for each subopcode. Routine BRSTAT uses register R10 as a pointer to the statistics table. When either R10 or the byte in the statistics table corresponding to the subopcode contains 0, no statistics information is logged.

Routine BRSTAT increments the offset count corresponding to the subopcode in the statistics table. If the count overflows, routine BRSTAT places > FF in location PDERR. DNOS monitors PDERR and outputs the statistics when PDERR contains > FF. After incrementing the count, BRSTAT uses the subopcode to obtain the subopcode processor address from the list shown in the calling sequence and returns to the DSR at the address of the subopcode processor.

5.5.2 End-of-Record Processor Routine

The DSR calls the end-of-record processor (ERSTAT or ENDRCD) to return the BRB to the I/O subsystem upon completion of the request. ERSTAT also collects information for online diagnostics.

Defined in Module

VOL.IOMGR.OBJECT.IONRCD

Entry

WS — PDT
R4 — Pointer to the PDT at PDTPDT
R10 — Pointer to the statistics table
PDTSRB — Pointer to the BRB

Calling Sequence

BL @ERSTAT or ENDRCD

Exit

R8 — modified
R9 — modified
R10 — modified

Routine ERSTAT uses R10 as a pointer to the statistics table. This is the same table used by BRSTAT. If register R10 contains a 0 (as when ENDRCD is called) or if the byte in the statistics table corresponding to the subopcode contains a 0, no statistics information is logged.

Routine ERSTAT gets the address of the BRB from the PDT, PDTSRB. It then masks interrupts to level 2, queues the processed request to the PDT, queues the PDT to the end-of-record list, and restores the interrupt level.

5.5.3 Bid Task Routine

The bid task routine (IOFCDT) bids a task from a DSR when a predefined pair of characters is entered at a keyboard device. System DSRs use this routine to implement the hard break key sequence (refer to Appendix A to see what keys this involves for the terminal you are using) and sequences beginning with the arming key (blank orange key). The supported sequences for the 911 VDT are as follows:

Blank orange, blank orange	Halt current output, resume current output
Blank orange, RETURN	Abort current output
Blank orange, CONTROL X	Terminate current task
Blank orange, (!)	Bid SCI

For the supported sequences, the arming character RESET is the blank orange key on the 911 VDT. You can specify the arming character in sequences required for your DSR. Routine IOFCDT requires that you specify a character for the function that aborts the current output. The routine first checks for an abort character. If the abort character is not found, the routine checks for a bid character defined for the device. You can define CDT entries to specify bidding additional tasks when other characters are entered.

An entry in the CDT defines the task to be bid from a DSR. The entry contains the LUNO assigned to the program file that contains the task to be bid and the ID of the task. It also contains the ASCII code of the character that bids the task. If the hard break key is allowed for this device, the CDT must contain an entry for the IOBREAK task (installed ID > 16 on the utility program file). The system requires a set of entries in a CDT for each device that can bid a task.

Defined in Module

VOL.IOMGR.OBJECT.IOKB

Entry

WS — KSB
R10 — Keyboard character (leftmost byte)

Calling Sequence

BL @IOFCDT
BYTE ASCII code of arming character
BYTE ASCII code of abort output character
DATA reserved
DATA alternate exit address

Exit

R6 — Modified
R9 — Modified
R10 — Modified

5.5.4 Queue Event Character or Queue Character Routine

The queue event character (PUTEBF) and the queue character (PUTCBF) routines store a character in the buffer to which the registers in the KSB point. If the queue is full, the character is not stored and the buffer full exit is taken.

Defined in Module

VOL.IOMGR.OBJECT.IOKB

Entry

WS — KSB
R1 — Count of characters in queue
R2 — Input queue pointer
R4 — Queue end pointer
R10 — Keyboard character (leftmost byte)

Calling Sequence

BL @PUTEBF or PUTCBF
DATA buffer full exit address

5.5.5 Get Queued Character Routine

The get queued character (GETC) routine removes a character from the character queue to which the registers in the KSB point and makes it available to the calling routine.

Defined in Module

VOL.IOMGR.OBJECT.IOKB

Entry

WS — PDT
R4 — Pointer to the PDT at PDTPDT

Calling Sequence

BL @GETC
DATA buffer empty exit address
DATA event character exit address

Exit

R0 — Modified
R6 — Buffer empty exit address
R9 — Queued character (leftmost byte)

If the queue is empty, the buffer empty exit address is placed into register R6 and an RTWP is executed. Otherwise, a character is removed from the queue. If the character is less than > 80, return is made to the calling routine at the instruction following the calling sequence.

If the character is > 9B, it is discarded and the next character is removed. If the character is in the range of > 80 through > 86 or > 96 through > 9F and the LUNO is not opened to accept event characters, the character is discarded and the next character is removed. Otherwise, the event flag is set in the system flags byte of the request and return is made via the event character address. When the request is a resource-independent call, the character is reinserted at the beginning of the character queue for future processing by a get event character routine.

5.5.6 Get Event Character

The get event character (IOGEC) routine removes an event character from the character queue to which the registers in the KSB point. If the buffer does not contain any event characters, no action is taken.

Defined in Module

VOL.IOMGR.OBJECT.IOKB

Entry

WS — PDT
R1 — Pointer to the BRB at the SVC opcode byte
R4 — Pointer to the PDT at PDTPDT

Calling Sequence

```
BL @IOGEC
```

Exit

```
R0 — Modified
R9 — Modified
R10 — Modified
```

The call to IOGEC should be added to subopcode processor 5. Routine IOGEC checks for the reply flag and extended request flag in the request call block. It then attempts to remove a character from the character queue to which the registers in the KSB point. If the character is in the range > 80 through > 86 or > 96 through > 9F and not > 9B, it is removed from the queue and replaced by > 9B. The character is placed in the event character byte of the request call block and the event flag is set in the system flags byte.

5.5.7 Character Check Routines

The seven-bit character check routine (ASCCHK) provides a means of checking for specific seven-bit characters. Likewise, the eight-bit character check routine (ASCCK2) provides a means of checking for specific eight-bit characters. Both routines transfer control to the corresponding routine when a specified character is found. A label can precede a call to ASCCHK.

Defined in Module

```
VOL.IOMGR.OBJECT.IOKB
```

Entry

```
WS — PDT
R9 — Character (msb)
```

Calling Sequence

```
BL @ASCCHK or ASCCK2
BYTE char,label — $ — 1/2
BYTE char,label — $ — 1/2
BYTE char,label — $ — 1/2
BYTE char,label — $ — 1/2
.
.
.
BYTE 0,0
```

Exit

```
R10 — Modified
```

5.5.8 Map Out Current Buffer Routine

The map out current buffer (IOMPOT) routine is used to retain the necessary information required to map a buffer. This routine allows a device to act upon more than one request at a time. The mapping information is stored in the request call block.

Defined in Module

`VOL.IOMGR.OBJECT.IOBMGT`

Entry

WS — PDT
R1 — Pointer to the BRB at the SVC opcode byte
R4 — Pointer to the PDT at PDTPDT

Calling Sequence

BL @IOMPOT

Exit

R0 — Modified

5.5.9 Get Buffer

The get buffer routine (IOGUB) obtains a buffer from the BTA (an example is a buffer for the PDT extension). You can access this buffer by using long-distance instructions. This routine is part of the root and does not require an include statement when linking.

Entry

WS — PDT
R1 — Buffer size
R10 — Pointer to a five-word temporary area

Calling Sequence

REF IOGUB
BL @IOGUB

Exit

R2 — BEET@

5.5.10 Map In Buffer

The map in buffer (IOMPIN) routine is used to map a buffer in if it has been mapped out with the map out buffer routine.

Defined in Module

`VOL.IOMGR.OBJECT.IOBMGT`

Entry

WS — PDT
 R1 — Pointer to the BRB at the SVC opcode byte
 R4 — Pointer to the PDT at PDTPDT

Calling Sequence

BL @IOMPIN

Exit

R0 — Modified

5.5.11 Get 20-Bit TILINE Address

The get 20-bit TILINE address (GTADDR) routine is used with TILINE devices to get a 21-bit physical byte address. The value is stored in the PDT extension defined by the DPD at DPDTIL > 10 to DPDTIL > 13.

Defined in Module

`VOL.IOMGR.OBJECT.IOTILN`

Entry

WS — PDT
 R1 — Pointer to the BRB at the SVC opcode byte

Calling Sequence

BL @GTADDR

Exit

R9 — Modified
 R10 — Modified

5.5.12 Transfer PDT Information to Task Memory

The transfer PDT information to task memory (XFERM) routine moves data in the PDT extension defined by the DPD at DPDWTK for the number of bytes specified.

Defined in Module

VOL.IOMGR.OBJECT.IOTILN

Entry

WS — PDT
R1 — Pointer to the BRB at the SVC opcode byte
R4 — Pointer to the PDT at PDTPDT
R6 — Count of characters to move

Calling Sequence

BL @XFERM

Exit

R6 — Modified
R9 — Modified
R10 — Modified

5.5.13 Report TILINE Error

The report TILINE error (TILERR) routine stores 16 bytes of TILINE information for the system log. If a previous error has been reported, the current error information is not stored.

Defined in Module

VOL.IOMGR.OBJECT.IOTILN

Entry

WS — PDT
R4 — Pointer to the PDT at PDTPDT
R10 — Error code (msb)
R12 — TPCS address

Calling Sequence

BL @TILERR

Exit

R8 — Modified
R9 — Modified

5.6 ASYNCHRONOUS DSR STRUCTURE

As the structure common to all DSRs has already been discussed, this paragraph describes the DSR structure specific only to asynchronous device support. In this paragraph, the term DSR refers to an asynchronous DSR. Table 5-2 shows the device and controller combinations that are supported by asynchronous DSRs.

Table 5-2. Asynchronous Device Support

Controllers	Devices					
	931	940	Business System Terminal	810	840	85X
CI401	Y	Y	Y			
CI421			Y	Y		Y
CI422			Y	Y		Y
/10A ¹	Y	Y	Y	Y		Y
CI402	Y	Y	Y	Y		Y
CI403	Y	Y	Y	Y		Y
CI404	Y	Y ²	Y ²	Y ²		Y ²
AUX1 of 931 ³				Y		Y
AUX1 of Business System Terminal ³				Y	Y	Y
AUX1 of 940 ³				Y	Y	Y

Notes:

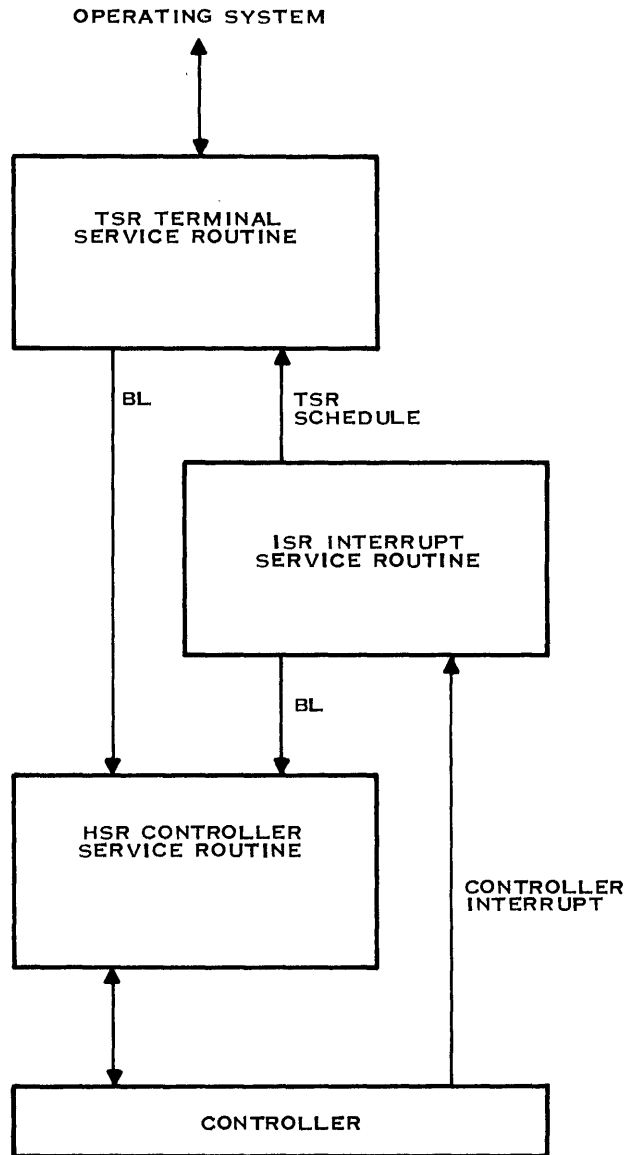
¹ /10A refers to the TMS9902 communications port on the 990/10A processor printed circuit board.

² These devices are connected to the CI404 via the fiber optics to EIA RS-232C converter module.

³ AUX1 refers to the auxiliary port found on the 931, 940, and Business System terminals.

The asynchronous DSR design separates controller and device support into different software modules. Figure 5-10 displays a block diagram of the DSR structure. The DSR consists of three basic modules. The hardware controller service routine (HSR) module provides the controller support. The terminal service routine (TSR) module provides device support. The interrupt service routine (ISR) module has interrupt and high priority processing responsibility. The following list describes the basic functions of the DSR components.

- **TSR module**
 - Contains all DSR entry points (hardware interrupt, system interrupt, power up, I/O abort, time-out, priority scheduler)
 - Provides request and completion reporting interface to DNOS
 - Runs in PDT workspace
 - Provides software interface to device
 - Contains device dependent logic
- **ISR module**
 - Contains hardware interrupt processing routine of the DSR
 - Provides interface to HSR for interrupt processing
 - Provides high priority receive character processing
 - Runs in DSR interrupt workspace (not the PDT workspace)
- **HSR module**
 - Provides generic (subroutine) software interface to the controller hardware
 - Contains all controller dependent logic
 - Contains all direct access to controller
 - Presents a buffered controller interface to other DSR modules



2284699

Figure 5-10. Asynchronous DSR Structure

5.6.1 Asynchronous DSR Design Overview

Figure 5-11 shows a detailed DSR flow diagram. This figure shows data flow paths as well as the DSR logic flow. Refer to Figure 5-11 during the following discussion of the DSR logic and data flow. The TSR module contains all DSR entry points. It accepts requests from, and reports completions to, the I/O subsystem of DNOS. The primary function of the TSR is to provide a software interface to the peripheral device. The actual functions vary considerably based on the type of device.

The TSR performs initial processing for all requests. The TSR calls the HSR for the output of data. The HSR stores output data in a transmit FIFO until the data can be transmitted on the communications line.

NOTE

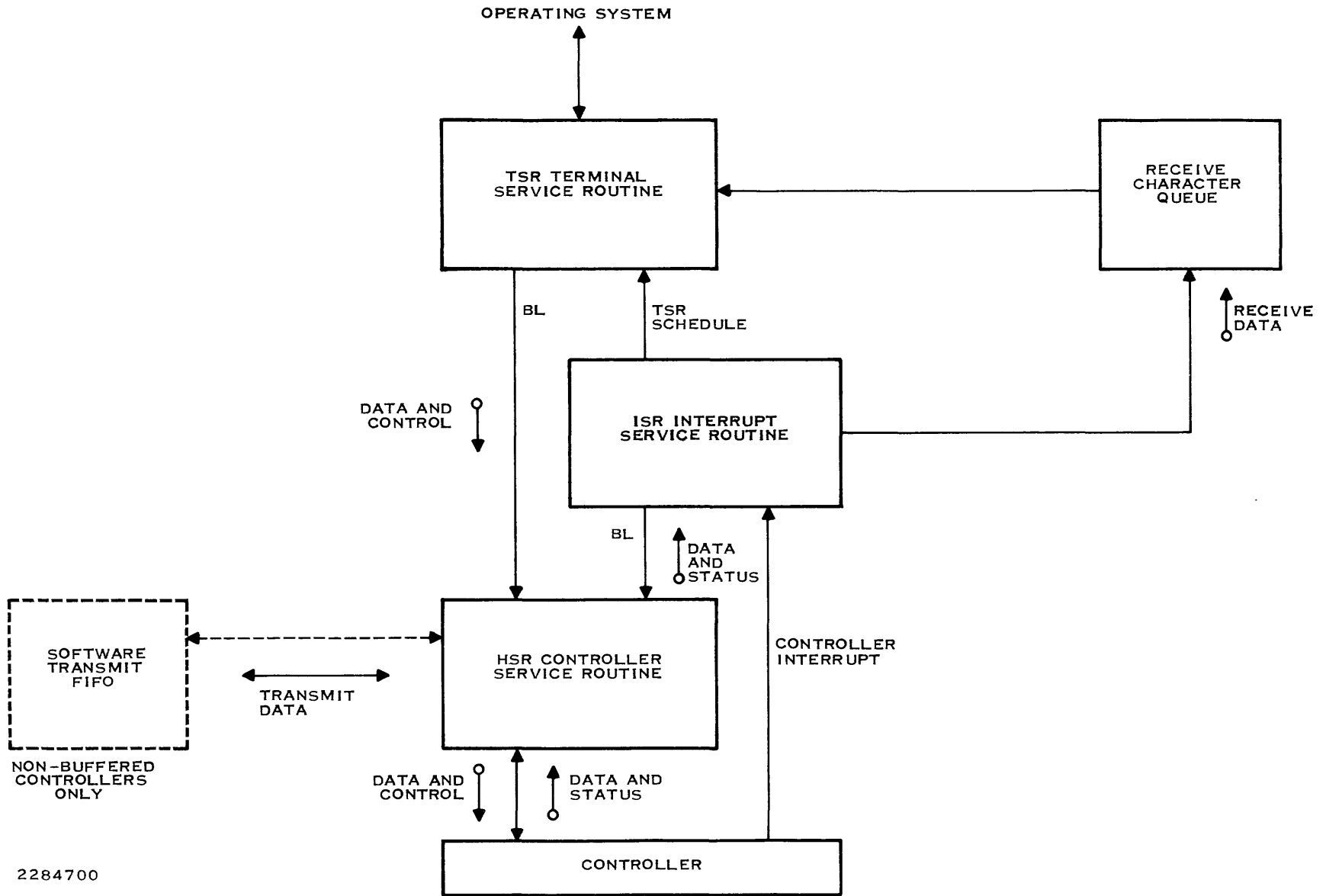
Buffered controllers such as the CI403 contain a hardware transmit FIFO. For non-buffered controllers such as the CI422, the HSR maintains a software transmit FIFO.

The HSR cannot accept data when the transmit FIFO is filled with data waiting to be transmitted. In this event, the TSR requests to be notified (by the ISR) when the HSR can accept more data. The HSR notifies the ISR when it can accept more transmit data, and the ISR schedules the TSR using the DSR priority schedule or reenter-me mechanism. The TSR can then resume transferring output data to the HSR. Figure 5-11 shows the logic paths followed in this process. Under normal conditions, the TSR reports completion of the output request before the HSR has actually transmitted all the data on the communications line.

Read requests, in most cases, require the cooperation of the TSR and ISR modules. The TSR attempts to satisfy the read by moving received data from the receive character queue to the user's read data buffer. When the TSR satisfies the read, it reports completion to the user task via the operating system I/O support routines. When there are not enough characters in the receive queue to satisfy the read, the TSR must wait. To do this, the TSR requests notification from the ISR when more data is stored in the receive character queue. The TSR then releases control. When the ISR stores data in the receive character queue, it schedules the TSR for execution. Other I/O service requests are processed by the TSR with the aid of the ISR when required.

The ISR module contains some functions that you can consider device support and some that you can consider controller support. The ISR module contains the hardware interrupt routine to which the TSR hardware interrupt entry points. The ISR module uses an interrupt workspace different than the physical device table (PDT) workspace. The ISR module runs with controller interrupts masked. The ISR calls the HSR to decode the controller interrupt.

For the most part, ISR processing is independent of request processing by the DSR. Received data is stored in the receive character queue even when no read request is active at the DSR. Error recovery action must be taken when the receive character queue becomes full. The ISR processes events requiring immediate attention. It also schedules the TSR module to start or resume processing.



2284700

Figure 5-11. Asynchronous DSR Logic Flow

The HSR provides access to the controller hardware. The HSR provides a generic interface to the controller. This allows other DSR modules to be written that are independent of the asynchronous controller type. HSR functions include controller and communications channel initialization, transmission of data, timer services, monitoring of modem signals, and controller interrupt decoding. The HSR does not support the concept of a read request. The HSR decodes the controller interrupt and reports the cause for the interrupt to the ISR. If the cause of the interrupt was a received data character, the data character is also passed back to the ISR. The HSR does not store the receive data.

5.6.2 Terminal Service Routine (TSR)

The TSR module is the interface to the I/O subsystem of DNOS. Like a DSR does, it must implement all the following DNOS interface functions:

- DNOS I/O subsystem entry points
 - Hardware interrupt
 - System interrupt (delayed reentry)
 - Power-up
 - Request abort
 - Request time-out
 - Priority scheduler
 - Request processor
- Data structures
 - Physical device table (PDT)
 - Keyboard status block (KSB)/interrupt workspace
 - Asynchronous device extensions (DSALLLEX, DSALLREX)
- I/O subsystem routines
 - BRSTAT
 - BZYCHK
 - ENDRCD
 - GETC
 - IOFCDT
 - IOFGEC

- IOGUB
- PUTCBF, PUTEBF

The following two mechanisms allow scheduling the TSR from an ISR:

- Delayed reentry
- Priority schedule

5.6.3 Interrupt Service Routine (ISR)

The interrupt service routine (ISR) contains the interrupt routine of the DSR and uses the ISR workspace of the DSR.

NOTE

Each channel of an interface supported by the asynchronous DSR structure must have an interrupt workspace different than the PDT workspace.

The ISR interfaces with both the TSR and the HSR. The design of the TSR/ISR interface is not dictated by the asynchronous DSR design. For the most part, you can specify it to fit your needs. The following list provides examples of ISR functions for standard keyboard devices:

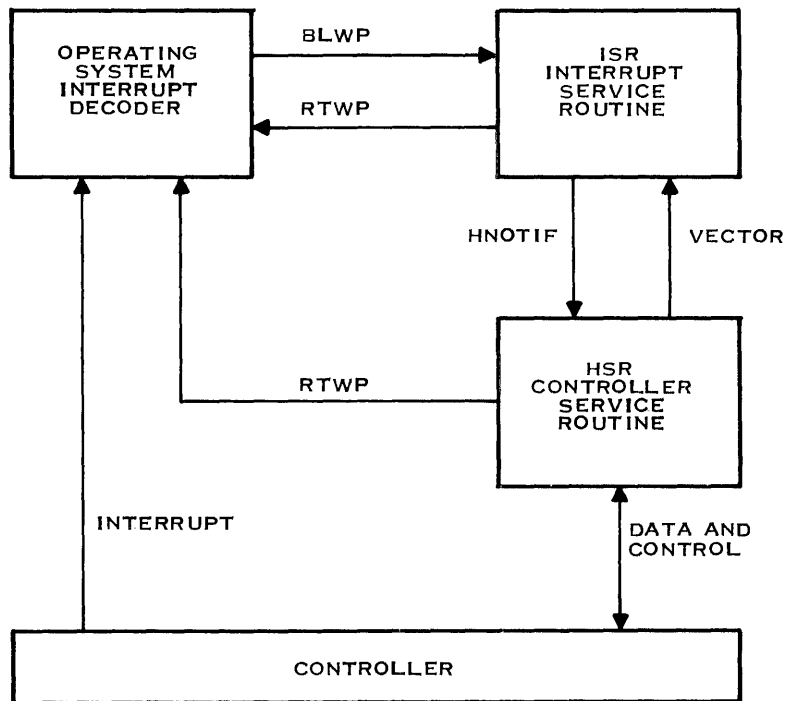
- Bid application task
- Suspend output
- Abort output
- Abort application task (hard break)

Figure 5-12 shows the flow of control during interrupt processing. The DSR is entered at its interrupt entry point via a BLWP instruction. The workspace upon entry is the DSR interrupt workspace. This is the same workspace the ISR uses.

The ISR is responsible for controlling further interrupt decoding by the DSR. If you are using the HSR supplied by Texas Instruments, the ISR calls the HSR subroutine HNOTIF (refer to paragraph 5.7.9) to determine what type of controller interrupt occurred. The ISR provides return vectors for each interrupting condition of the controller. Figure 5-12 uses the word VECTOR to describe this process. The description of the HNOTIF subroutine documents the set of generic interrupt conditions. The HNOTIF subroutine takes the return vector associated with the current controller interrupt.

The ISR code for the specific type of interrupt takes the proper action to service the interrupt. When complete, the ISR returns to the operating system interrupt decoder via an RTWP instruction. The operating system decoder takes the necessary action to restore normal system execution.

Figure 5-12 shows one other path for interrupt processing. The HSR exits or returns directly to the operating system interrupt decoder if HNOTIF is called when no controller interrupt is pending. This special return exists to support certain ISRs; most user ISRs never use this path.



2284701

Figure 5-12. Interrupt Processing Flow

5.6.4 Hardware Controller Service Routine (HSR)

The generic interface to the HSR consists of a set of subroutines with a branch and link (BL) call interface. A subroutine implements one or more generic functions for the specific controller in use. For example, the TSR makes a Set DTR subroutine call. The HSR for a CRU controller might implement this as a SBO DTR CRU instruction. However, the HSR for a TILINE controller may implement the same subroutine by using a SOC @DTR,@OUTSIG(R12) instruction to access the TILINE Peripheral Control Space (TPCS) of the controller. Identical requests from the TSR/ISR invoke identical functions for all controllers. Provision is made for controller hardware differences. A “not supported” return is provided for most HSR subroutines. This return is taken when the requested function is not supported by the controller hardware.

The asynchronous DSR design can support several device and controller combinations. HSR object modules are provided with the operating system for the controllers listed in Table 5-3. Table 5-3 also documents the final node in the pathname for each HSR. The directory that contains the HSR object modules is < volume> .S\$OSLINK.DEVDSR.OBJECT, where < volume> is a synonym that you assign.

Table 5-3. HSR Object Modules

Name	Controller Type
DS403HSR	CI403/CI404
DS923HSR	TMS 9902 and 9903 controllers (*)
DS401HSR	CI401 (previously COMM I/F)
Note:	
* Includes CI402, CI422, CI421, the 990/10A 9902 port, and the 9902 interface associated with the internal terminal on the Business System 300 computer.	

These modules are made available for users implementing DSRs for special devices connected to these controllers. These HSR modules are used for DSRs following the asynchronous DSR design. This design must be followed for user written DSRs under the following conditions:

- The special device is connected to a CI403 or CI404 controller
- A standard TI DSR is used to support any of the communication channels of the CI403 or CI404

When these conditions do not occur, you have a choice of DSR designs for asynchronous device support. You can implement either the asynchronous DSR design or a design of your choice. Remember that any user design must observe all DNOS constraints.

5.6.5 Asynchronous Data Structure Allocation

The DIB (the PDT extension) is divided into two segments. One segment is physically contiguous to the PDT. This segment contains data that requires the most frequent access and must be created before the ALGS step of system generation is done. The segment must have the following pathname:

```
< volume> .S$OSLINK.S$SGU$.SD.DIBXXXX
```

The other segment contains data for which an increased access time does not significantly affect overall performance. The second segment must be accessed using long-distance instructions. Other data structures included in the long-distance extension are VDT screen images for VDT DSRs. Refer to Figure 5-13.

All the data structures that are not accessed via long-distance instructions must be created before the completion of system generation. These data structures are available when the operating system is loaded into memory from disk. The long-distance data structures must be allocated during the power-up initialization of the operating system. The TSR is responsible for allocating the second long distance PDT extension by using the system routine IOGUB. If you are using the HSR module supplied by Texas Instruments, this extension must be allocated and words PDXSMB and PDXSMP must be initialized with the PDT extension address before the TSR can call any of the HSR subroutines. The size of the extension must be at least 144 bytes for nonbuffered controllers and at least 32 bytes for buffered controllers (CI403/CI404). The user has the option of specifying additional space for the user-written TSR/ISR. The user can also lengthen the local PDT extension by reserving more memory in the DIB that is provided to the system generation program.

DNOS DATA STRUCTURES

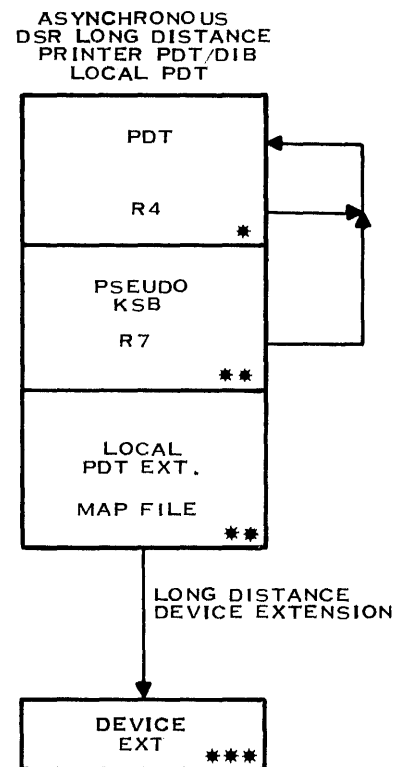
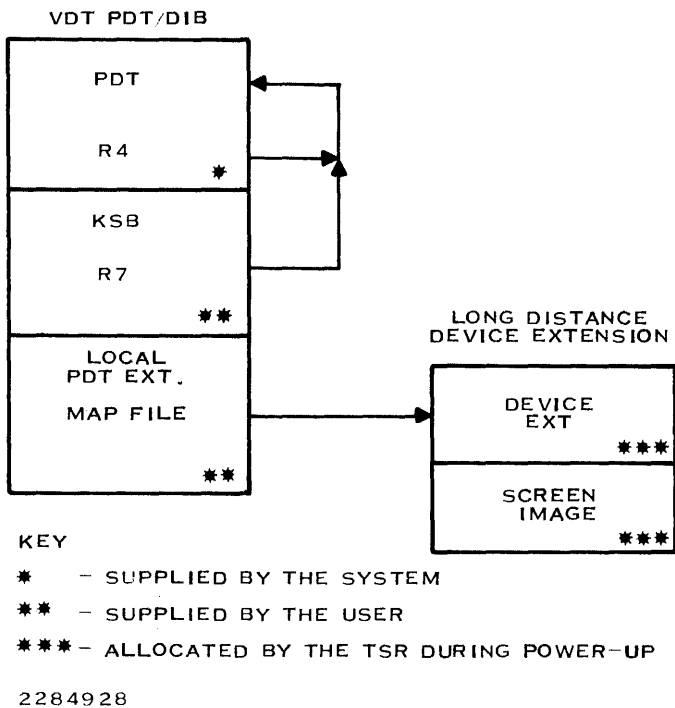


Figure 5-13. Asynchronous Data Structure Linkages

5.7 HSR COMMON SUBROUTINES

The information required to interface to the HSR module is as follows:

- Subroutine names
- Functions provided by each subroutine
- Subroutine calling conventions

This information is discussed in the following paragraphs.

The following list describes the HSR subroutine classes. Each class contains several subroutines. These subroutines provide one or more HSR functions.

- Power-up initialization
- Write output signal or function
- Read input signal or function
- Enable/disable status change notification
- Output a character
- Write operational parameters
- Read operational parameters
- Request timer interval notification
- Controller interrupt decoding

All HSR subroutines are called via branch and link (BL) instructions. Thus, they use the caller's workspace during execution. Parameters required by the subroutines are passed to the HSR in workspace registers. Information is returned to the caller in one of two ways. Data is returned to the caller in workspace registers. Other status information is returned by way of alternate subroutine returns. The caller specifies alternate return addresses as operands of DATA assembler directives immediately following the BL subroutine call. The following shows an example of HSR alternate return addresses:

```

BL    @HSRSUB
DATA ALT1
DATA ALT2
****
SUBROUTINE CALL
FIRST ALTERNATE RETURN
SECOND ALTERNATE RETURN
NORMAL RETURN (CODE)

```

The caller execution resumes at one of the alternate return addresses or at the normal return address (the instruction following all alternate return DATA statements). The number of alternate returns varies for different HSR subroutines.

The HSR subroutines follow some general register conventions. The subroutines normally use R0 and R10 as working registers. These two registers are also used when parameters are passed to or from the HSR. Exceptions are noted in some of the HSR subroutine descriptions. In most cases, R7 is used as a pointer to the PDT. R12 contains the TILINE or CRU base address of the controller or controller channel. Other register usage is documented with specific HSR subroutines.

5.7.1 Power-Up Initialization

This subroutine class allows the HSR to perform any initialization required before operation begins. The long distance buffer must be allocated before the HSR power-up subroutine is called.

For the CI403 and CI404, the HSR is required to ensure that the controller has successfully executed the self-test for each channel specified during system generation. For other controllers, the HSR may be required to build a software transmit FIFO in a long-distance memory buffer that is obtained by the operating system.

The three subroutines that perform power-up initialization functions are as follows: HRESET, HSWPWR, and HMRST.

The HRESET subroutine performs power-up initialization that must be performed for all I/O channels of the controller. This subroutine can be called once per channel for multiple channel controllers. However, there is only one master reset of the controller for each power-up occurrence.

The HSWPWR subroutine performs all channel-oriented initialization. The HSR data structures for the channel are initialized. Controller interrupts are enabled when the normal return is taken.

The HMRST subroutine performs the same initialization functions as the HRESET routine except that a master reset of the controller is unconditionally performed. This subroutine is provided so that diagnostic software can force a controller master reset for testing purposes.

The TSR normally makes two calls to the HSR for power-up initialization. The HRESET subroutine is called first, followed by the HSWPWR subroutine. These two subroutines are called for each channel of the controller. The calling conventions are identical for each of the HSR power-up subroutines. If HPOWER is considered a synonym for any of the three power-up subroutine names, then the calling conventions for all HSR power-up subroutines are as follows:

Calling convention:

BL	@HPOWER	
DATA	XXXX	POWER UP FAILURE RETURN VECTOR
****		NORMAL RETURN (CODE)

5.7.2 Write Output Signal or Function

The subroutine names for setting output signals or functions to 1 (logic true) are of the form HSTxxx, where xxx specifies a signal or function. The subroutine names for resetting output signals or functions to 0 (logic false) are of the form HRTxxx, where xxx specifies a signal or function. The following list describes the output signals supported by HSRs:

Subroutine	Signal
HSTAL, HRTAL	AL — Analog Loopback (Signal)
HSTDTR, HRTDTR	DTR — Data Terminal Ready (Signal)
HSTRTS, HTRTS	RTS — Request to Send (Signal)
HSTSRS, HTRSRS	DSRS — Data Signal Rate Select (Signal)
HSTSRT, HTRSRT	SRTS — Secondary Request to Send (or Reverse Channel Signal)

The following list describes the output functions supported by HSRs.

Subroutine	Function
HSTBIL, HRTBIL	BIL — Board Internal Loopback (Function)
HSTCR, HRTCR	CR — Channel Reset (Function)
HSTCTH, HRTCTH	CTH — Channel Transmitter Halt (Function)
HSTRS, HTRRS	RS — Receiver Squelch (Function)
HSTTB, HRTTB	TB — Set Transmit Break condition (Function)
HSTUIL, HRTUIL	UIL — UART Internal Loopback (Function)

The calling conventions for the HSTxxx and HRTxxx subroutines are identical. The following calling convention uses the HSTxxx name as an example:

Calling convention:

BL	@HSTxxx	WHERE xxx IS DTR, RTS, ETC.
DATA	VVVV	SIGNAL NOT SUPPORTED RETURN VECTOR
----		NORMAL EXIT (CODE)

The following paragraphs describe each of the functions in detail.

5.7.2.1 HSTBIL Subroutine. The Set Board Internal Loopback subroutine (HSTBIL) provides for setting a more general (controller or board) internal loopback mode than the UART internal loopback mode. None of the current asynchronous controllers support this second level of loopback.

5.7.2.2 HSTCR Subroutine. The Set Channel Reset subroutine (HSTCR) performs a hardware reset of the channel. This does not disturb any other communication channels. The channel interrupts are not enabled by this subroutine. The HRTCR subroutine enables interrupts and allows normal operation. The following example shows a typical sequence for TSR use of these subroutines:

1. TSR issues an HSTCR call to quiet HSR activity on the channel.
2. TSR issues an HSWPWR call to initialize the HSR data structures and status.

3. TSR initializes its channel data structures and status.
4. TSR then issues an HRTCR call to begin normal operation.

5.7.2.3 HSTCTH Subroutine. The Set Channel Transmitter Halt subroutine (HSTCTH) temporarily suspends output of data to the communications line. Any data in the transmit FIFO is not transmitted. The HSTCTH subroutine resumes transmission of data in the transmit FIFO.

5.7.2.4 HSTRS Subroutine. The Set Receiver Squelch subroutine (HSTRS) enables half-duplex operation. The receiver squelch function disables reception of data during transmission. The HSTRS subroutine turns the receiver squelch off and allows full-duplex operation.

5.7.2.5 HSTTB Subroutine. The Set Transmit Break subroutine (HSTTB) initiates the transmission of a break sequence (spacing/logic 0). This continues until stopped with the HRTTB subroutine.

5.7.2.6 HSTUIL Subroutine. The Set UART Internal Loopback subroutine (HSTUIL) places the UART (communications chip) for the channel in loopback mode. In general, this causes the UART to return all transmitted data as received data on the same channel. Refer to UART documentation for more detailed information. The HRTUIL subroutine changes the UART from UART internal loopback to normal mode.

5.7.3 Read Input Signal or Function

This subroutine class allows reading controller input signals and HSR function states. The HSR subroutine name is of the form HRDxxx, where xxx identifies a signal or function. The following list describes the input signals or function states that can be read:

Subroutine	Signal/Function
HRDBIL	BIL — Board Internal Loopback (Signal)
HRDCR	CR — Channel Reset (Function)
HRDCTH	CTH — Channel Transmission Halted (Function)
HRDCTS	CTS — Clear to Send (Signal)
HRDDCD	DCD — Data Carrier Detect (Signal)
HRDDSR	DSR — Data Set Ready (Signal)
HRDRI	RI — Ring Indicator (Signal)
HRDSCT	SCTS — Secondary Clear to Send (Signal)
HRSDCD	SDCD — Secondary Data Carrier Detect (or Speed Indication Signal)
HRDSSS	SSS — Split Speed Supported (Function)
HRDTB	TB — Transmit Break (Function)
HRDUIL	UIL — UART Internal Loopback (Signal)

Calling Convention:

BL	@HRDxxx	WHERE xxx IS DSR, CTS, ETC.
DATA	}}	SIGNAL NOT SUPPORTED RETURN VECTOR
DATA	YYYY	CONTROLLER FAILURE RETURN VECTOR
DATA	ZZZZ	SIGNAL FALSE RETURN VECTOR
----		SIGNAL TRUE RETURN (CODE)

5.7.4 Enable/Disable Status Change Notification

This subroutine class provides a mechanism for the ISR (and therefore, indirectly the TSR) to receive status change notification from the HSR. There are subroutines to enable notification and to disable notification. Once enabled, most of the signals or functions that are supported remain enabled until explicitly disabled.

The Transmit Shift Register Empty (TSRE) function is an exception to this rule. If the HSR makes a TSRE status notification, it automatically disables further notification for this same condition. The TSRE function must be enabled with another HDSTSR call to the HSR if subsequent notification is desired. Refer to the HSR interrupt decoder description for more details about the method of notification.

Notification is made when the signal changes from 0 to 1 or from 1 to 0 for the first five signals. Some controllers notify only on the ring signal changing from 0 to 1. The TSRE notification is made only when the condition occurs.

The subroutine names for enabling status change notification are of the form HESxxx, where xxx specifies a signal or function. The subroutine names for disabling status change notification are of the form HDSxxx, where xxx specifies a signal or function. The following list describes the notification conditions that the HSR supports:

Subroutine	Status Change Notification
HESCTS, HDSCTS	CTS — Clear to Send
HESDCD, HSDSCD	DCD — Data Carrier Detect
HESDSR, HSDSR	DSR — Data Set Ready
HESRI, HDSRI	RI — Ring Indicator
HES SCT, HDSSCT	SCTS — Secondary Clear to Send
HES SDC, HDSSDC	SDCD — Secondary Data Carrier Detect
HES TSR, HDSTSR	TSRE — Transmit Shift Register Empty

The calling conventions for the HESxxx and HDSxxx subroutines are identical. The following calling convention uses the HESxxx name as an example:

Calling Convention:

BL	@HESxxx	WHERE xxx IS DSR, RI, ETC.
DATA	VVVV	NOT SUPPORTED RETURN VECTOR
----		NORMAL RETURN (CODE)

5.7.5 Output a Character

This HSR subroutine accepts characters to be output on the communications channel. The subroutine provides a character interface to the output channel (that is, only one character is passed to the HSR for each HOUTPx subroutine call). In all cases, the output data is stored in a transmit FIFO before transmission on the communications line. For buffered controllers, the FIFO is on the hardware controller. For non-buffered controllers, the FIFO is a software data structure that the HSR manages. An alternate (character not output) return from the HOUTPx routine is taken if the FIFO becomes full. The caller is responsible for saving the data character. The HSR notifies the ISR when the transmit FIFO is empty. This notification takes place as a transmit interrupt exit from the HSR interrupt decoder subroutine. This notification causes the output data to flow to the HSR again. Refer to the HSR interrupt decoder subroutine description for more details.

The output character subroutines are HOUTP4 and HOUTP7. The only difference in these two subroutines is that workspace register R4 contains the PDT pointer for the HOUTP4 subroutine and R7 contains the PDT pointer for the HOUTP7 subroutine.

Calling Convention:

BL	@HOUTPx	x = 4 IF R4 IS PDT POINTER
		x = 7 IF R7 IS PDT POINTER
DATA	XXXX	CHARACTER NOT OUTPUT — RETURN VECTOR
----		NORMAL (CHARACTER OUTPUT) RETURN

where:

R7 or R4	contains the pointer to the PDT.
R5	is the output character, left byte.

Volatile Registers:

R0 and R5
R5 preserved for character not output

5.7.6 Write Operational Parameters

The following list describes the operational parameters that the HSRs support.

- Baud rate selection — The transmit baud rate and the receive baud rate are specified in the most significant bit (MSB) and least significant bit (LSB) of R0, respectively. The MSB and LSB must be identical for controllers not supporting dual speeds.
- Data format:
 - Parity selection: even, odd, mark, space, or none
 - Character length selection: 5, 6, 7, or 8 bit data
 - Stop bit selection: 1, 1.5, 2 stop bits

The write parameters subroutines are Set Channel Speed (Baud Rate) and Set Data Character Format. The calling conventions for these two subroutines are very similar. The only difference is the parameter information passed in R0.

5.7.6.1 Set Channel Speed (Baud Rate). This subroutine specifies the transmit and receive rates for the channel. The transmit and receive rates may differ only when dual rates are supported.

Calling Convention:

```

BL    @HSPSPD          SET CHANNEL SPEED
DATA  VVVV            PARAMETER NOT SUPPORTED RETURN VECTOR
-----              NORMAL RETURN (CODE)
    
```

where:

R0 MSB contains the transmit rate code;
 LSB contains the receive rate code.
 (Refer to Table 5-4.)

Table 5-4. HSR Baud Rate Codes

Speed Code	Baud Rate
00	50 baud
01	75 baud
02	110 baud
03	134.5 baud
04	150 baud
05	200 baud
06	300 baud
07	600 baud
08	1,200 baud
09	1,800 baud
0A	2,400 baud
0B	3,600 baud
0C	4,800 baud
0D	7,200 baud
0E	9,600 baud
0F	14,400 baud
10	19,200 baud
11 through FF	Reserved

5.7.6.2 Set Data Character Format. This subroutine sets the character length, parity selection, and the number of stop bits for data characters.

Calling Convention:

BL @HSPPSL	SET DATA CHARACTER FORMAT PARAMETERS
DATA VVVV	PARAMETER NOT SUPPORTED RETURN VECTOR
----	NORMAL RETURN (CODE)

where:

R0 contains the parameter information in the following format:

Bit	Contents
0 – 1	Reserved
2 – 3	Parity selection 00 = odd parity 01 = even parity 10 = mark parity 11 = space parity
4	Parity enable 00 = not enabled 01 = enabled
5 – 7	Reserved
8 – 9	Number of stop bits 00 = 1 stop bit 01 = 1.5 stop bits 10 = reserved 11 = 2 stop bits
10 – 11	Data character length 00 = 5 bit character 01 = 6 bit character 10 = 7 bit character 11 = 8 bit character
12 – 15	Reserved

5.7.7 Read Operational Parameters and Information

This class of subroutines allows the following operational parameter values to be read from the HSR:

Subroutine	Operational Parameter Value
HRPDAT	HSR module revision level
HRPPSL	Data format: Parity selection: even, odd, mark, space, or none Character length selection
HRPSPD	Stop bit selection
HRPTYP	Baud rate Controller type ID

The parameter information is returned in R0 of the caller's workspace. For the HRPSPD and HRPPSL subroutines, the format of the information in R0 is identical to the format in the HSPSPD and HSPPSL subroutines, respectively.

Calling Convention:

```
BL    @HRPxxx          WHERE xxx IS SPD OR PSL
-----                RETURN
```

The HRPDAT subroutine returns the current revision level of the HSR software module in R0. The revision level is a hexadecimal number starting at 0 for the initial level and incrementing by one for each revision. The HRPTYP subroutine returns a code right justified in R0 that identifies the controller type. Table 5-5 lists controller type codes.

Table 5-5. Controller Type Codes

Code	Controller
> 0001	CI401 (previously COMM I/F)
> 0006	Business System 300 internal 9902 port
> 0007	990/10A 9902 port
> 0008	CI402
> 0009	CI421 9902 port
> 000A	CI422
> 0023	CI403
> 0024	CI404
> 0030	CI421 9903 port

5.7.8 Request Time Interval Notification

This subroutine (HTIMER) requests notification after a specified time interval. You specify the time interval as a multiple of 250-millisecond periods. The HSR interrupt decoder performs the notification by taking the timer interrupt vector return to the ISR. Refer to the discussion of the HSR interrupt decoder for more details. Timer notification is disabled by specifying a zero as the number of 250 millisecond intervals (R0 = 0). There is not a "not supported" exit provided for this subroutine.

Calling Convention:

```
BL    @HTIMER
```

where:

R0 specifies the number of 250-millisecond intervals.

5.7.9 Controller Interrupt Decoder

The ISR calls this subroutine (HNOTIF) to perform controller interrupt decoding. The subroutine executes in the DSR interrupt workspace and with interrupts masked to the interrupt level of the controller channel. The ISR provides several return vector addresses via DATA directives immediately following the call. A return vector is provided for each interrupt type possible from the controller. If the subroutine finds no controller interrupt pending, the return is to the operating system interrupt decoder rather than the caller.

Calling Convention:

BL	@HNOTIF	
DATA	XXXX	RECEIVE INTERRUPT VECTOR
DATA	YYYY	TRANSMIT INTERRUPT VECTOR
DATA	ZZZZ	SIGNAL OR FUNCTION CHANGE VECTOR
DATA	AAAA	TIMER INTERRUPT VECTOR
DATA	BBBB	ILLEGAL/INVALID INTERRUPT VECTOR

Receive Interrupt Return:

R10 Received character left byte
 Line status in right byte

Bit	Meaning
0-2	Reserved
3	Break Received
4	Framing Error
5	Parity Error
6	Overrun Error
7	Reserved

Transmit Interrupt Return:

This return is taken when the transmit FIFO empties.

Signal or Function Change Return:

R10 Current signal or function states are returned in bits 0 – 3 and bits 8 – 10. Bits 4 – 7 and 12 – 15 are delta flags that indicate which signals or functions changed.

Bit	Contents
0	DCD
1	RI
2	DSR
3	CTS
4	Delta DCD
5	Delta RI
6	Delta DSR
7	Delta CTS
8	SCTS
9	SDCD
10	TSRE
11	Reserved
12	Delta SCTS
13	Delta SDCD
14	Delta TSRE
15	Reserved

5.8 DSR INSTALLATION

After writing the DSR, you must assemble and link edit it. Assemble the DSR by executing the Execute Macro Assembler (XMA) SCi command and making the correct responses.

After you assemble the DSR, you must link it with all of the required support subroutines. This is required with each release of the operating system, not with each system generation between releases. The following example shows a typical link control stream used to link a DSR:

```

NOPAGE
ERROR
PROCEDURE  DUMROOT
DUMMY
INCLUDE    <volume>.$OSLINK.$SGU$.DUMROOT
PHASE     0,DSRname,PROG >C000
INCLUDE    dsr object pathname
INCLUDE    <volume>.$OSLINK.IOMGR.OBJECT.IONRCD
INCLUDE    (any other support routines)
END

```

Note: < volume> refers to the volume name of the data disk used during system generation.

The Execute Link Editor (XLE) SCI command executes the Link Editor. The linked output access name should be <volume>.\$OSLINK.\$SGU\$.SD.DSRxxxxx, where xxxxx represents the special device name defined at system generation. Later, the system generation utility will expect the DSR to be found in this file.

Special extensions to the PDT (DIBs) must be in the file <volume>.\$OSLINK.\$SGU\$.SD.DIBxxyy, where xx is the first two characters of the special device name specified during system generation (ST if the device has a keyboard), and yy is the device ID generated by the system generation utility (for example, 03 in ST03).

If you build an asynchronous DSR that is to be linked with an HSR module that is supplied by TI, you must first include the TSR module (which contains the DSR entry points). The following is an example of a link control stream:

```

NOPAGE
ERROR
PROCEDURE    DUMROOT
DUMMY
INCLUDE      <volume>.$OSLINK.$SGU$.DUMROOT
PHASE        0,DSRname,PROG >C000
INCLUDE      tsr object pathname
INCLUDE      <volume>.$OSLINK.DEVD$R.OBJECT.HSRname
INCLUDE      <volume>.$OSLINK.IOMGR.OBJECT.IONRCD
INCLUDE      (any other support routines)
END
    
```

Note: <volume> refers to the volume name of the data disk used during system generation.

5.9 DEBUGGING TECHNIQUES

The best debugging technique is to produce a well-documented DSR. This makes it easier to locate errors and makes the coding clearer to others. In addition to good documentation, thorough code reading with colleagues helps reduce errors.

Missing indexing registers are typical of the errors found through code reading. When the structure begins with IRBxxx, R1 should be the index register. When the structure begins with PDTxxx or is an extension to the PDT, R4 should be the index register.

The next step is debugging the DSR on the computer in a restricted environment. The JMP \$ instruction can be placed at strategic points within the DSR. When the computer executes one of these instructions, the PC remains at that location until the JMP \$ instruction is removed. This enables the programmer to use the computer front panel to examine DSR behavior and data used by the DSR. The SIE mode can be used if the value of ST is changed to xxx2.

5.10 DSR EXAMPLE

Figure 5-14 shows the source code listing of a DSR for a line printer. This listing shows a typical application of the guidelines for writing a DSR. You may adapt this DSR to nonsupported devices.


```

SDSMAC 3.5.0 82.130 16:53:03 FRIDAY, JUN 17, 1983.
ACCESS NAMES TABLE
SOURCE ACCESS NAME= DS01.DSRLP
OBJECT ACCESS NAME= DUMY
LISTING ACCESS NAME= DS01.LPLST
ERROR ACCESS NAME=
OPTIONS= RXREF
MACRO LIBRARY PATHNAME=
LINE KEY NAME
0002 A DSC.CONDASM.OS
=>.$SOSLINK.CONDASM.OS
0140 LI DSC.MACROS.TEMPLATE
=>.$SOSLINK.MACROS.TEMPLATE
0141 LI DSC.MACROS.FUNC
=>.$SOSLINK.MACROS.FUNC
0149 B DSC.TEMPLATE.ATABLE.IRB
=>.$SOSLINK.TEMPLATE.ATABLE.IRB
0150 C DSC.TEMPLATE.ATABLE.PDT
=>.$SOSLINK.TEMPLATE.ATABLE.PDT
0157 D DSC.TEMPLATE.ATABLE.LPD
=>.$SOSLINK.TEMPLATE.ATABLE.LPD
0256 E DSC.TEMPLATE.COMMON.NFERR1
=>.$SOSLINK.TEMPLATE.COMMON.NFERR1
E0007 F DSC.TEMPLATE.COMMON.NFER00
=>.$SOSLINK.TEMPLATE.COMMON.NFER00
E0008 G DSC.TEMPLATE.COMMON.NFER10
=>.$SOSLINK.TEMPLATE.COMMON.NFER10
E0009 H DSC.TEMPLATE.COMMON.NFER20
=>.$SOSLINK.TEMPLATE.COMMON.NFER20
E0010 I DSC.TEMPLATE.COMMON.NFER30
=>.$SOSLINK.TEMPLATE.COMMON.NFER30
0257 J DSC.TEMPLATE.COMMON.NFWORD
=>.$SOSLINK.TEMPLATE.COMMON.NFWORD
DSRLP SDSMAC 3.5.0 82.130 16:53:03 FRIDAY, JUN 17, 1983.
DSRLP - DNOS LINE PRINTER DSR
0002 COPY DSC.CONDASM.OS
0006 IDT 'DSRLP'
0016 *
0017 * (C) COPYRIGHT, TEXAS INSTRUMENTS INCORPORATED, 1979.
0018 * ALL RIGHTS RESERVED. PROPERTY OF TEXAS INSTRUMENTS
0019 * INCORPORATED. RESTRICTED RIGHTS - USE, DUPLICATION
0020 * OR DISCLOSURE IS SUBJECT TO RESTRICTIONS SET FORTH
0021 * IN TI'S PROGRAM LICENSE AGREEMENT AND ASSOCIATED
0022 * DOCUMENTATION.
0023 *
0024 * ROUTINE NAME: DSRLP
0025 *
0026 * ABSTRACT:
0030 * FOR LPDIFF => 0
0043 * FOR LPDIFF < 0
0051 * THIS IS THE HANDLER FOR I/O TO VARIOUS MODELS
0052 * OF THE CENTRONICS AND TI LINE PRINTERS, THAT
0053 * USE AN RS-232-C INTERFACE.
0054 *
0055 * ENTRY: ENTERED VIA THE OPERATING SYSTEM THROUGH
0056 * DIFFERENT ENTRY POINTS.
0057 *
0058 * EXIT: VIA AN RTWP
0059 *
0060 * ERRORS: >02 - ILLEGAL OPCODE
0061 * >04 - POWER LOST
0062 * >06 - REQUEST ABORTED
0063 *
0064 * REVISION: 07/18/80 - ORIGINAL
0065 * <REVISION DATE; LATEST LAST> - <NATURE>
0066 *
0067 * REVISION 02/04/81 - HANDLE ''READ ONLY'' 840,820 TERMINALS.
0068 * MAKE THIS DSR SMART ENOUGH TO RECOGNIZE
0069 * DC3 (BUSY) & DC1 (READY) SIGNALS.
0070 * 01 07/30/81 -DNOS 1.1
0071 * ADD 9902 CONTROLLER COMMUNICATION
0072 *

```

Figure 5-14. DSR Listing Example (Sheet 1 of 16)

```

0073      *          09/14/81 - ADD CONDITIONAL ASSEMBLIES FOR DX10
0074      *
0075      *    02    01/6/82 - CHANGE 9902 POWER UP TO SET 9902 FOR
0076      *          A 2.5 MHz CLK
0077      *    03    02/12/82 - FIX ALGORITHM AS TO WHEN IT IS SAFE
0078      *          TO CALL ENDRCD.
0079      *    04    03/22/82 - DON'T DO LINE FEED ON OPEN
0080      *    05    03/22/82 - BE SURE EOR FLAG IS RESET WHEN A FORCE
0081      *          ENDRCD EXIT IS MADE.
0082      *    06    03/23/82 - ABORT PROCESS SHOULD SET EOR FLAG IF
0083      *    07    10/01/82 - ADD 9902 2-CHANNEL MUX SUPPORT
0084      *          PDTSRB IS NON-ZERO.
0092      *    09    02/25/83 - CORRECT SLOW DOWN OF PARALLEL INTERFAC
0093      *          PRINTERS.
0094      *    10    03/22/83 - 9902 GETS PROGRAMMABLE BAUD RATE
0095      *    11    03/28/83 - FIX BUGS
0096      *    12    04/08/83 - ABORT I/O CODE HAS BUGS
0097      *    13    04/21/83 - READST RETURNS STATISTICS
0098      *    14    06/03/83 - Fix lower case on parallel printer bug
0099      *
0100      * ENVIRONMENT: 990/10 ASSEMBLER
0101      *          CALLABLE FROM ASSEMBLER
0102      *          TABLE SEGMENTS MAPPED IN WHEN ENTERED:
DSRRLP      SDSMAC 3.5.0 82.130    16:53:03 FRIDAY, JUN 17, 1983.
DSRRLP - DNOS LINE PRINTER DSR          PAGE 0003
0103      *          <STA, BTA, DSR CODE>
0104      *          TABLE SEGMENTS MAPPED IN DURING ROUTINE:
0105      *          <NONE>
0106      *
0107      *
0108      * SUBROUTINE REFS:
0109      *          REF ENDRCD          END-OF-RECORD ROUTINE
0110      *          REF BRSTAT          BRANCH TABLE PROCESSOR
0111      *
0112      *
0113      * MACROS TO BE USED:
0114      *          LIBIN DSC.MACROS.TEMPLATE          =
0115      *          LIBIN DSC.MACROS.FUNC              =
0116      *
0117      * EQUATES:
0118      *          ASMIF C$OS=C$DPOS = DNOS ONLY =====
0119      *          COPY DSC.TEMPLATE.ATABLE.PDT      =
0120      *          COPY DSC.TEMPLATE.ATABLE.IRB      =
0121      *          COPY DSC.TEMPLATE.ATABLE.PDT      =
0122      *          0066 LPDBGN EQU PDTsiz            =
0123      *          COPY DSC.TEMPLATE.ATABLE.LPD      =
0124      *          ASMEND =====
0125      *          1 ASMIF C$OS=C$DX10 = DX10 ONLY =====
0126      *          UNL
0127      *          COPY DSC.SYSTEM.TABLES.DSRPDT     =
0128      *          COPY DSC.SYSTEM.TABLES.LPD        =
0129      *          COPY DSC.SYSTEM.TABLES.DSRIRB     =
0130      *          LIST
0131      *          COPY DSC.SYSTEM.TABLES.DSRPDT     =
0132      *          COPY DSC.SYSTEM.TABLES.LPD        =
0133      *          COPY DSC.SYSTEM.TABLES.DSRPDT     =
0134      *          LPDQCC EQU LPDCC                  =
0135      *          LPDQIP EQU LPDIN                  =
0136      *          LPDQOP EQU LPDOUT                 =
0137      *          LPDIFF EQU LPDDMF                 =
0138      *          ASMEND =====
DSRRLP      SDSMAC 3.5.0 82.130    16:53:03 FRIDAY, JUN 17, 1983.
DSRRLP - DNOS LINE PRINTER DSR          PAGE 0004
0139      * +-----+
0140      *          EIA CRU INPUT BIT DEFINITIONS
0141      * +-----+
0142      *          >0          INPUT DATA (LSB)
0143      *          >7          INPUT DATA (MSB)
0144      *          0008 EIAxMT EQU >8          TRANSMIT IN PROGRESS (0=NO; 1=YES)
0145      *          0009 EIAERR EQU >9          TIMING ERROR (0=NO; 1=YES)
0146      *          >A          REVERSE CHANNEL RECv (0=NO; 1=YES)
0147      *          >B          WRITE REQUEST (0=NO; 1=YES)
0148      *          >C          READ REQUEST (0=NO; 1=YES)

```

Figure 5-14. DSR Listing Example (Sheet 2 of 16)

```

0186      000D EIADCD EQU >D      DATA CARRIER DETECT (0=NO; 1=YES)
0187      000E EIADSR EQU >E      DATA SET READY (0=NO; 1=YES)
0188      * >F      MODULE INTERRUPT (0=NO; 1=YES)
0189      * -----
0190      * EIA CRU OUTPUT DEFINITIONS
0191      * -----
0192      * >0      DATA TO MODULE (LSB)
0193      0007 EIAPAR EQU >7      DATA TO MODULE (MSB, PARITY BIT)
0194      * >8      * NOT USED *
0195      0009 EIADTR EQU >9      DATA TERMINAL READY (0=OFF; 1=ON)
0196      000A EIARTS EQU >A      REQUEST TO SEND (0=OFF; 1=ON)
0197      000B EIAWRQ EQU >B      WRITE REQUEST (CLEAR) (0/1 CLEARS )
0198      000C EIARRQ EQU >C      READ REQUEST (CLEAR) (0/1 CLEARS )
0199      000D EIANSF EQU >D      NEW STATUS FLAG(CLEAR)(0/1 CLEARS )
0200      000E EIAIEN EQU >E      INTERRUPT ENABLE (0=OFF; 1=ON)
0201      000F EIADIM EQU >F      DIAGNOSTICS MODE (0=OFF; 1=ON)
0202      * -----
0203      *
0204      *
0205      *
0206      *
0207      * -----
0208      * DATA MODULE(DM) CRU OUTPUT BIT DEFINITIONS
0209      * -----
0210      * >0      DATA TO MODULE (LSB)
0211      * >6      DATA TO MODULE (MSB)
0212      0007 DMSTR EQU >7      DATA STROBE (ASCII)
0213      0008 DMSTRJ EQU >8      DATA STROBE (JISCI)
0214      0009 DMVFC EQU >9      VERTICAL FROMS CONTROL
0215      * >A
0216      * >B
0217      * >C
0218      000D DMDDM EQU >D      DEMAND FOR A CHARACTER
0219      000E DMINT EQU >E      INTERRUPT ENABLE BIT
0220      000F DMCIN EQU >F      INTERRUPT ACKNOWLEDGE BIT
0221      * -----
0222      * 9902 CRU INPUT BIT DEFINITIONS R01
0223      * ----- R01
0224      *
0225      0022 SECDCD EQU >22      SECONDARY DATA CARRIER DETECT R01
0226      0020 DCD902 EQU >20      DATA CARRIER DETECT R01
0227      001B DSR902 EQU >1B      DATA SET READY R01
0228      0016 XBRE EQU >16      XMIT BUFR REGISTER EMPTY R01
0229      0010 RRQ902 EQU >10      READ REQUEST INTERRUPT R01
0230      *
0231      * ----- R01
0232      * 9902 CRU OUTPUT BIT DEFINITIONS R01
0233      * ----- R01
0234      *
0235      0027 ABLTRS EQU >27      ENABLE TRANSMISSIOS R01
DSRLP    SDSMAC 3.5.0 82.130 16:53:03 FRIDAY, JUN 17, 1983.
DSRLP - DNOS LINE PRINTER DSR PAGE 0005
0236      0026 INT902 EQU >26      INTERRUPT ENABLE R01
0237      0024 CLOCK EQU >24      1 = 2MHZ R06
0238      * 0 = 4MHZ R06
0239      0022 SECRTS EQU >22      SECONDARY RTS R01
0240      0020 DTR902 EQU >20      DATA TERMINAL READY R01
0241      001F RESET EQU >1F      RESET CONTROLLER R01
0242      0015 DSCENB EQU >15      DATA SET STATUS CHANGE R01
0243      * INTERRUPT ENABLED R01
0244      0013 XBIENB EQU >13      XMIT BUFR REGISTER EMPTY R01
0245      0012 RIENB EQU >12      RCV'R INT ENABLE
0246      * INTERRUPT ENABLED R01
0247      0010 RTS902 EQU >10      REQUEST TO SEND ON R01
0248      * ----- R01
0252      0003 PR9902 EQU 3 LPD FLAG R01=
0253      * BIT 3 =1 -> 9902 CONTROLLED R01=
0254      * GLOBAL DATA: =
E0008    COPY DSC.TEMPLATE.COMMON.NFER10
E0009    COPY DSC.TEMPLATE.COMMON.NFER20
E0010    COPY DSC.TEMPLATE.COMMON.NFER30
0257    COPY DSC.TEMPLATE.COMMON.NFWORD =

```

Figure 5-14. DSR Listing Example (Sheet 3 of 16)

```

0259      *      COPY DSC.TEMPLATE.COMMON.NFERR1      =
0260      *      COPY DSC.TEMPLATE.COMMON.NFWORD      =
0261      *      ASMEND =====
0262      *
0263      *      NOTES:
0264      *
DSRLP      SDSMAC 3.5.0 82.130 16:53:03 FRIDAY, JUN 17, 1983.
DSRLP      - DNOS LINE PRINTER DSR                      PAGE 0006
0266      *
0267      *      ROUTINE NAME: DSRLP
0268      *
0272      *      ABSTRACT: THIS IS THE MAIN ENTRY POINT INTO THE DSR FOR
0273      *      THE HARDWARE INTERRUPT (LPINT), SYSTEM INTERRUPT
0274      *      (LPINT), POWER RESTORE (PWRON), ABORT I/O
0275      *      ROUTINE, (ABORT), REQUEST TIME OUT (ABORT), AND
0276      *      REQUEST PROCESSING. CERTAIN INITIALIZATION IS
0277      *      PERFORMED PRIOR TO PASSING CONTROL TO THE OP
0278      *      CODE HANDLING ROUTINE VIA 'BRSTAT'.
0279      *
0280      0000 0460      B      @LPINT      DSR HARDWARE INTERRUPT
0281      0002 02C4'
0281      0004 0460      B      @LPSINT      DSR SOFTWARE INTERRUPT
0281      0006 02CA'
0282      0008 0460      B      @PWRON      DSR POWER UP
0282      000A 033C'
0283      000C 0460      B      @ABORT      DSR ABORT
0283      000E 0326'
0284      0010 0460      B      @ABORT      DSR TIME OUT
0284      0012 0326'
0307      0014 04C5      CLR R5      INITIALIZE COUNTER
0308      0016 0207      LI R7,PAGE1     INITIALIZE BUFFER
0308      0018 0072'
DSRLP      SDSMAC 3.5.0 82.130 16:53:03 FRIDAY, JUN 17, 1983.
DSRLP      - DNOS LINE PRINTER DSR                      PAGE 0007
0310      001A 020A      LI R10,STAB     POINT TO STATISTICS TABLE
0310      001C 004E'
0311      001E 06A0      BL @BRSTAT      DECODE OPCODE AND BRANCH
0311      0020 0000
0312      *-----
0313      *      THE STATISTICS TABLE MUST BE MODIFIED IF THE OPCODE
0314      *      TABLE IS MODIFIED.
0315      *-----
0316      0022 0013      BASE DATA MAXCOD      MAX OP CODE
0317      0024 014E'      DATA ILLOP      ILLEGAL OP RETURN
0318      0026 007C'      DATA OPEN      0 OPEN
0319      0028 007A'      DATA CLOSE     1 CLOSE
0320      002A 007E'      DATA REWIND    2 CLOSE EOF
0321      002C 0080'      DATA OPNRWD  3 OPEN REWIND
0322      002E 007E'      DATA REWIND    4 CLOSE UNLOAD
0323      0030 008A'      DATA READST    5 READ STATUS
0324      0032 014E'      DATA EXIT1    6 FWD SPACE - IGNORE      R05
0325      0034 014E'      DATA EXIT1    7 BAK SPACE - IGNORE      R05
0326      0036 014E'      DATA ILLOP    8 UNUSED - ILLEGAL
0327      0038 014E'      DATA ILLOP    9 READ ASCII - ILLEGAL
0328      003A 014E'      DATA ILLOP   A READ BINARY- ILLEGAL
0329      003C 0062'      DATA WRITE    B WRITE ASCII
0330      003E 0062'      DATA WRITE    C WRITE DIRECT
0331      0040 007E'      DATA REWIND    D WRITE EOF
0332      0042 007E'      DATA REWIND    E REWIND
0333      0044 014E'      DATA EXIT1    F UNLOAD - IGNORE      R05
0334      0046 014E'      DATA ILLOP   10 UNUSED - ILLEGAL
0335      0048 014E'      DATA ILLOP   11 UNUSED - ILLEGAL
0336      004A 014E'      DATA ILLOP   12 UNUSED - ILLEGAL
0337      004C 013C'      DATA DUMP     13 DUMP STATISTICS
0338      0013 MAXCOD EQU ($-BASE-6)/2
0339      004E 48      STAB BYTE PDTMC      0 OPEN
0340      004F 48      BYTE PDTMC      1 CLOSE
0341      0050 48      BYTE PDTMC      2 CLOSE EOF
0342      0051 48      BYTE PDTMC      3 OPEN REWIND
0343      0052 48      BYTE PDTMC      4 CLOSE UNLOAD
0344      0053 00      BYTE 0      5 READ STATUS
0345      0054 00      BYTE 0      6 FWD SPACE

```

Figure 5-14. DSR Listing Example (Sheet 4 of 16)

Writing a DSR

```

00B4 010C'
0404 00B6 0208      LI  R8,>0500      ASSUME EIA INTERFACE      R13
00B8 0500
DSRLP  SDSMAC 3.5.0 82.130 16:53:03 FRIDAY, JUN 17, 1983.
DSRLP  - DNOS LINE PRINTER DSR      PAGE 0009
0405 00BA C164      MOV  @LPDIFF(R4),R5      GET INTERFACE FLAGS      R13
00BC 0066
0406 00BE 2160      COC  @LPF902*2+MASTAB,R5  IS THIS A 9902 ?      R13
00C0 0028+
0407 00C2 1602      JNE  RDST10      IF NOT, SKIP      R13
0408 00C4 0208      LI  R8,>0600      SET UP FOR A 9902 INTRFCE R13
00C6 0600
0409 00C8      RDST10 EVEN      R13
0419 00C8 2560      CZC  @LPFIF*2+MASTAB,R5  IS THIS PARALLEL ?  = R13
00CA 0022+
0420 00CC 1602      JNE  RDST20      IF NOT, SKIP      = R13
0424 00CE 0208      LI  R8,>8000      SET UP FOR PARALLEL      R13
00D0 8000
0425 00D2 06A0      RDST20 BL  @RFILL1      ONE WORD TO STORE      R13
00D4 010C'
0426 00D6 04C8      CLR  R8      6 WORDS OF ZEROS      R13
0427 00D8 0205      LI  R5,6      R13
00DA 0006
0428 00DC 06A0      BL  @RFILL      R13
00DE 0110'
0429 00E0 D224      MOVB @LPDSPX(R4),R8      GET XMIT BAUD RATE      R13
00E2 0076
0430 00E4 06A0      BL  @RFILL1      STORE IN OUTPUT BUFFER  R13
00E6 010C'
0431 00E8 04C8      CLR  R8      R13
0432 00EA 0205      LI  R5,12      12 WORDS OF ZEROS      R13
00EC 000C
0433 00EE 06A0      BL  @RFILL      R13
00F0 0110'
0434 00F2 C204      MOV  R4,R8      GET PDT POINTER      R13
0435 00F4 0228      AI  R8,PDTRC      POINT AT READ COUNT      R13
00F6 0044
0436 00F8 0205      LI  R5,6      6 BYTES TO COPY      R13
00FA 0006
0437 00FC 06A0      BL  @RCOPY      R13
00FE 0126'
0438 0100 04C8      CLR  R8      FINALLY, FOUR MORE ZEROS R13
0439 0102 0205      LI  R5,4      R13
0104 0004
0440 0106 06A0      BL  @RFILL      R13
0108 0110'
0441 010A 1021      JMP  EXIT1      R03
0442      *      R13
0443      *      TO AID IN THE CONSTRUCTION OF THE STATUS RETURN R13
0444      *      BUFFER, WE HAVE RFILL AND RCOPY. R7 S JOB IS TO R13
0445      *      POINT AT THE OUTPUT BUFFER.      R13
0446      *      RFILL - STORE R8 INTO BUFFER R5 TIMES      R13
0447      *      RCOPY - COPY R5 BYTES FROM *R8 TO BUFFER      R13
0448      *      SHOULD THE BUFFER BECOME FULL AT ANY TIME, RFILL R13
0449      *      AND RCOPY TAKE THE LIBERTY TO TERMINATE THE SVC R13
0450      *      AND RETURN WITH WHATEVER INFORMATION MADE IT INTO R13
0451      *      THE BUFFER.      R13
0452      *      R13
0453 010C 0205      RFILL1 LI  R5,1      FOR ONE WORD STORES      R13
010E 0001
0454 0110 8861      RFILL C  @IRBOCC(R1),@IRBICC(R1) BUFFER FULL ?      R13
0112 000A
0114 0008
0455 0116 1516      JGT  EXIT      IF SO, BLOW THIS      R13
0456 0118 1315      JEQ  EXIT      POPSICLE STAND      R13
DSRLP  SDSMAC 3.5.0 82.130 16:53:03 FRIDAY, JUN 17, 1983.
DSRLP  - DNOS LINE PRINTER DSR      PAGE 0010
0457 011A CDC8      MOV  R8,*R7+      ELSE, DO MORE FILLING  R13
0458 011C 05E1      INCT @IRBOCC(R1)      KICK COUNTER HARD      R13
011E 000A
0459 0120 0605      DEC  R5      DONE YET ?      R13
0460 0122 16F6      JNE  RFILL      IF NOT, HIT IT AGAIN  R13

```

Figure 5-14. DSR Listing Example (Sheet 6 of 16)


```

0534 0164 1309      JEQ  TSTDSR          -YES.                R01
0535 0166 1D13      SBO  XBIENB          R01
0536 0168 1D10      SBO  RTS902          R01
0537 016A 0460      B    @TST480        -NO. EXIT.. WAIT    R01
016C 0270'
0538                *
0539 016E 1F0B      TSTW05 TB  EIAWRQ      WRITE REQUEST HIGH?  R01
0540 0170 1302      JEQ  TSTWR1          MAR
0541 0172 0460      B    @TST480        NO, WAIT FOR IT     MAR
0174 0270'
0542 0176 1E0B      TSTWR1 SBZ  EIAWRQ      YES, RESET IT
0543                *
0547 0178 C2A4      TSTDSR MOV  @LPDQEP(R4),R10  GET QUEUE END POINTER  =
017A 006E
0558 017C C264      TSTRR9 MOV  @LPDQIP(R4),R9   GET QUEUE INPUT POINTER
017E 006A
DSRLP  SDSMAC 3.5.0 82.130  16:53:03 FRIDAY, JUN 17, 1983.
DSRLP  - DNOS LINE PRINTER DSR          PAGE 0013
0560                *
0561                *  BUFFER THE DATA IF POSSIBLE
0562                *
0563 0180 C145      MOV  R5,R5          ANY DATA TO BUFFER?
0564 0182 130E      JEQ  TST030          NO
0565 0184 824A      TST010 C  R10,R9     YES, AT END OF BUFFER?
0566 0186 1B01      JH   TST020          NO
0570 0188 625A      S    *R10,R9        YES, PUT POINTER AT BEGINNING=
0571 018A 86A4      TST020 C  @LPDQCC(R4),*R10  IS THE BUFFER FULL?  =
018C 0068
0582 018E 1308      JEQ  TST030          YES
0583 0190 DE77      MOVB *R7+,*R9+      NO, MOVE NEXT CHAR TO BUFFER
0584 0192 05A4      INC  @LPDQCC(R4)    ...COUNT IT
0194 0068
0585 0196 0605      DEC  R5              ...REDUCE INPUT COUNT
0586 0198 15F5      JGT  TST010
0587                *
0588                *  ALL THE DATA IS BUFFERED  R03
0589                *  SET END-OF-RECORD FLAG.    R03
0593 019A E920      SOC  @LPFEOR*2+MASTAB,@LPDIFF(R4)  ENDRCD CAN NOW BE CALLED.  R03
019C 002A+
019E 0066
DSRLP  SDSMAC 3.5.0 82.130  16:53:03 FRIDAY, JUN 17, 1983.
DSRLP  - DNOS LINE PRINTER DSR          PAGE 0014
0603 01A0 C909      TST030 MOV  R9,@LPDQIP(R4)  SAVE QUEUE INPUT POINTER
01A2 006A
0604 01A4 0206      LI   R6,LPSPUR      INITIALIZE INTERRUPT VECTOR
01A6 03A0'
0605 01A8 C224      MOV  @LPDQCC(R4),R8  ANY DATA TO OUTPUT?
01AA 0068
0606 01AC 1602      JNE  TST035          R01
0607 01AE 0460      B    @TST480        NO                    R01
01B0 0270'
0608 01B2 C224      TST035 MOV  @LPDQOP(R4),R8  YES, GET QUEUE OUTPUT POINTER
01B4 006C
0609 01B6 820A      C    R10,R8         AT END OF BUFFER?
0610 01B8 1B01      JH   TST040          NO
0614 01BA 621A      S    *R10,R8        YES, POINTER TO BEGINNING  =
0623 01BC 0206      TST040 LI   R6,TSTDSR    INITIALIZE INTERRUPT VECTOR
01BE 0178'
0627 01C0 C024      MOV  @LPDIFF(R4),R0  DM INTERFACED LP?
01C2 0066
0628 01C4 1505      JGT  DMOUT          -YES.                R01
0629 01C6 1304      JEQ  DMOUT          -YES.                R01
0640                *
0641 01C8 2020      COC  @LPF902*2+MASTAB,R0
01CA 0028+
0642                *
0643 01CC 1328      JEQ  OUT902          IS THIS 9902 CONTROLLED  R01
0644 01CE 103C      JMP  EIAOUT         -YES. JUMP           R01
0644 01CE 103C      JMP  EIAOUT         -NO. MUST BE EIA      R01
DSRLP  SDSMAC 3.5.0 82.130  16:53:03 FRIDAY, JUN 17, 1983.
DSRLP  - DNOS LINE PRINTER DSR          PAGE 0015
0646                *-----
0647                *  OUTPUT TO DATA MODULE (DM) INTERFACE

```

Figure 5-14. DSR Listing Example (Sheet 8 of 16)


```

0648
0649      01D0  DMOUT EQU $                                09
0650 01D0 1F0D TB DMDMD                                DEMAND UP?
0651 01D2 134E JEQ TST480                                NO, GO EXIT DSR
0652 01D4 0278 MOVB *R8+,R9                                YES, PICKUP NEXT OUTPUT CHAR
0653 01D6 0A20 SLA R0,LPFUC+1                            MAP LOWERCASE TO UPPERCASE?
0654 01D8 1808 JOC TST210                                NO
0655 01DA 0289 CI R9,>6100                                YES
          01DC 6100
0656 01DE 1A05 JL TST210
0657 01E0 0289 CI R9,>7B00
          01E2 7B00
0658 01E4 1402 JHE TST210
0659 01E6 0249 ANDI R9,#>2000
          01E8 DFFF
0660 01EA 0549 TST210 INV R9                                INVERT THE CHARACTER
0661 01EC 20A0 COC @DSFJIS*2+MASTAB,R2                    FE
          01EE 002A+
0662      *
0663 01F0 1605 JNE TST220                                IS IT JISCI terminal?    FE
0664 01F2 3209 LDCR R9,8                                YES, OUTPUT 8 BITS
0665 01F4 1E08 SBZ DMSTRJ                                STROBE THE INTERFACE
0666 01F6 1000 NOP
0667 01F8 1D08 SBO DMSTRJ
0668 01FA 1004 JMP TST240
0669      *
0670 01FC 31C9 TST220 LDCR R9,7                                NO, OUTPUT 7 BITS
0671 01FE 1E07 TST230 SBZ DMSTR                                STROBE THE INTERFACE
0672 0200 1000 NOP
0673 0202 1D07 SBO DMSTR
0674 0204 C908 TST240 MOV R8,@LPDQOP(R4)                UPDATE OUTPUT POINTER
          0206 006C
0675 0208 0624 DEC @LPDQCC(R4)                            REDUCE QUEUE OUTPUT COUNT
          020A 0068
0676 020C 13B5 JEQ TSTDSR                                GO BUFFER MORE DATA    09
0677 020E C024 MOV @LPDIFF(R4),R0                            Grab interface flags    14
          0210 0066
0678 0212 8288 C R8,R10                                AT END OF BUFFER?      09
0679 0214 16DD JNE DMOUT                                NO-CONTINUE PRINTING   09
0683 0216 621A S *R10,R8                                YES, RESET POINTER     09
0692 0218 C908 MOV R8,@LPDQOP(R4)                            UPDATE OUTPUT POINTER   09
          021A 006C
0693 021C 10D9 JMP DMOUT                                MOVE NEXT CHAR         09
DSRLP SDSMAC 3.5.0 82.130 16:53:03 FRIDAY, JUN 17, 1983.
DSRLP - DNOS LINE PRINTER DSR                                PAGE 0016
0695      *-----
0696      * OUTPUT TO 9902 INTERFACE
0697      *-----
0698      *
0699      *
          * IS PRINTER BUSY?
0700 021E 2020 OUT902 COC @LPFBSY*2+MASTAB,R0
          0220 0026+
0701 0222 1302 JEQ OUT100                                R01
0702 0224 1F1B TB DSR902                                DATA SET READY?      R01
0703 0226 1303 JEQ OUT200                                R01
0704 0228 1E13 OUT100 SBZ XBIENB                            -NO. DISABLE XMIT BUFF INT R01
0705 022A 1D15 SBO DSCENB                                ENABLE DAT SET READY INT R01
0706 022C 1021 JMP TST480                                R01
0707 022E C28B OUT200 MOV R11,R10                            R01
0708 0230 1D13 SBO XBIENB
0709 0232 06A0 BL @LPCOMN                                R01
          0234 0274
0710 0236 C2CA MOV R10,R11
0711 0238 0249 OUT410 ANDI R9,>7F00                            R01
          023A 7F00
0712      *
0713 023C 3209 LDCR R9,8                                SEND THE CHARACTER     R01
0714 023E C908 MOV R8,@LPDQOP(R4)                            UPDATE OUTPUT POINTER   R01
          0240 006C
0715 0242 0624 DEC @LPDQCC(R4)                            REDUCE QUEUE CHAR COUNT R01
          0244 0068
0716 0246 1014 JMP TST480                                EXIT                    R01

```

Figure 5-14. DSR Listing Example (Sheet 9 of 16)

```

0717          *-----
0718          * OUTPUT TO EIA INTERFACE
0719          *-----
0720          *
0721          *
0722          *
0723 0248 2020 EIAOUT COC @LFPBSY*2+MASTAB,R0
          024A 0026+
0724 024C 1311 JEQ TST480 YES
0725 024E 1F0E TB EIAADR ''DATA SET READY'' ON ??
0726 0250 160F JNE TST480 NO
0727 0252 C28B MOV R11,R10
0728 0254 06A0 BL @LPCOMN
          0256 0274'
0729 0258 C2CA MOV R10,R11
0730 025A 0249 ANDI R9,>7F00
          025C 7F00
0731 025E 31C9 LDCR R9,7 LOAD CHARACTER
0732 0260 1C02 JOP TST440 SET PARITY BIT AS NEEDED
0733 0262 1E07 SBZ EIAPAR
0734 0264 1001 JMP TST450
0735 0266 1D07 TST440 SBO EIAPAR
0736 0268 C908 TST450 MOV R8,@LPDQOP(R4) UPDATE OUTPUT POINTER
          026A 006C
0737 026C 0624 DEC @LPDQCC(R4) REDUCE QUEUE CHARACTER COUNT
          026E 0068
0738 0270 0460 TST480 B @EXIT PERFORM END OF REQUEST
          0272 0144'
DSRLP SDSMAC 3.5.0 82.130 16:53:03 FRIDAY, JUN 17, 1983.
DSRLP - DNOS LINE PRINTER DSR PAGE 0017
0740          *
0741          *-----
0742          * LPCOMN - COMMON ROUTINE USED BY 9902 AND EIA PRINTERS
0743          *
0744          * 1) PROCESS EXTENDED CHARACTER SET.. IF ANY
0745          * 2) CHECK AND PROCESS KATAKANA
          *-----
0746 0274 0206 LPCOMN LI R6,TSTWRQ SET INTERRUPT VECTOR
          0276 015A'
0747 0278 D278 MOVB *R8+,R9 NEXT OUTPUT CHAR
0748 027A 0A20 SLA R0,LPFUC+1 EXTENDED CHARACTER SET
0749 027C 1808 JOC LPC410 -YES. JUMP
0750 027E 0289 CI R9,>6100 -NO. MAP LOWERCASE TO
          0280 6100
0751 0282 1A05 JL LPC410 UPPERCASE?
0752 0284 0289 CI R9,>7B00
          0286 7B00
0753 0288 1402 JHE LPC410
0754 028A 0249 ANDI R9,#>2000
          028C DFFF
0755          *
0756 028E 20A0 LPC410 COC @DSFJIS*2+MASTAB,R2 IS THIS A JISCII TERMINAL
          0290 002A+
0757 0292 1617 JNE LPC430
0784 0294 D249 MOVB R9,R9 KATAKANA CHARACTER? =
0785 0296 110B JLT LPC420 YES =
0786          * PRINTER IN ALPHA MODE =
0787 0298 20A0 COC @DSFJAR*2+MASTAB,R2 =
          029A 002E+
0788 029C 1612 JNE LPC430 YES =
0789 029E 0242 ANDI R2,#(>8000//DSFJAR) NO, RESET TO ALPHA =
          02A0 FFFF
0790 02A2 0209 LI R9,>0F00 LOAD SHIFT IN CODE =
          02A4 0F00
0791 02A6 0608 DEC R8 RESET QUEUE OUTPUT POINTER =
0792 02A8 05A4 INC @LPDQCC(R4) ADJUST QCC =
          02AA 0068
0793 02AC 100A JMP LPC430 =
0794          * =
0795          * PRINTER IN KATAKANA MODE? =
0796 02AE 20A0 LPC420 COC @DSFJAR*2+MASTAB,R2 =
          02B0 002E+
0797 02B2 1307 JEQ LPC430 YES =

```

Figure 5-14. DSR Listing Example (Sheet 10 of 16)


```

0872 0300 9800 LPI030 CB R0,@DC1 A ''READY'' INTERRUPT ? FE
      0302 0388'
0873 0304 1603 JNE LPI040 NO. JUMP FE
0874 * YES. RESET BUSY BIT FE
0875 0306 4920 SZC @LPFBSY*2+MASTAB,@LPDIFF(R4) MA
      0308 0026+
      030A 0066
0876 030C C024 LPI040 MOV @LPDIFF(R4),R0 9902 CONTROLLED ? R01
      030E 0066
0877 0310 0A40 SLA R0,@LPF902+1 R01
0878 0312 1703 JNC LPI045 -NO. JUMP R01
0879 0314 1F1B TB DSR902 -YES. DSR STILL ON? R01
0880 0316 1306 JEQ LPI050 -YES. JUMP R01
0881 0318 1002 JMP LPI047 R01
0882 031A 1F0E LPI045 TB EIADSR ''DATA SET READY'' ON? R01
0883 031C 1303 JEQ LPI050 YES. FE
0884 * NO. TURN OFF BUSY FLAG FE
0885 * USE ''DSR'' SIGNAL AS BUSY FE
0886 * INDICATOR. FE
0887 031E 4920 LPI047 SZC @LPFBSY*2+MASTAB,@LPDIFF(R4) R01
      0320 0026+
      0322 0066
0888 0324 0456 LPI050 B *R6 FE
0889 *
0890 0326 C164 ABORT MOV @PDTSRB(R4),R5 ENDRCD REQUIRED? R06
      0328 005E
0891 032A 1306 JEQ ABRT10 -NO. R06
0892 * -YES. SET EOR FLAG
0903 032C E920 SOC @LPFEOR*2+MASTAB,@LPDIFF(R4) R06=
      032E 002A+
      0330 0066
0907 0332 E920 SOC @DFGOPF*2+MASTAB,@PDFTLG(R4) SET OP-FAILED R12
      0334 002C+
      0336 000E
0908 0338 04C5 ABRT10 CLR R5 CLEAR COUNT TO BUFFER R06
0909 033A 0456 B *R6 PROCESS NEXT CHAR
DSRLP SDSMAC 3.5.0 82.130 16:53:03 FRIDAY, JUN 17, 1983.
DSRLP - DNOS LINE PRINTER DSR PAGE 0020
0921 033C C024 PWRON MOV @LPDIFF(R4),R0 DM INTERFACED LP ?
      033E 0066
0922 0340 1107 JLT PWR1 NO
0926 0342 1E0F SBZ DMCIN YES, CLEAR INPUT INTERRUPTS
0927 0344 1D0E SBO DMINT ENABLE INTERRUPTS
0928 0346 1D09 SBO DMVFC DISABLE VFC
0929 0348 1D07 SBO DMSTR INITIALIZE STROBE (ASCII)
0930 034A 1D08 SBO DMSTRJ INITIALIZE STROBE (JISCI)
0931 034C 1024 JMP PWR2 MA
0932 034E 4600 INIT DATA >4600 (EIAIEN/EIARTS/EIADTR)
0933 0350 0A40 PWR1 SLA R0,LPF902+1 9902 CONTROLLED? R01
0934 0352 171D JNC PWR120 R01
0935 0354 1D1F SBO RESET RESET THE 9902 CONTROLLER R01
0936 0356 1F08 TB 8 CONTROLLER EXIST ?? R06
0937 0358 132C JEQ NOTBZY -NO. R06
0938 035A 0208 LI R8,CNTL25 ASSUME A 2.5 MHZ 9902 R10
      035C A200
0939 035E 0209 LI R9,BR25S2 R10
      0360 03BA'
0940 0362 1F24 TB CLOCK IS THIS A 2.5MHZ 9902 R06
0941 0364 1304 JEQ PWR110 -YES
0942 0366 0208 LI R8,CNTL4 -NO. 4MHZ. GET CONTROL R06
      0368 AA00
0943 036A 0209 LI R9,BR40S2 GET BUAD RATE R06
      036C 03E0'
0944 036E 3208 PWR110 LDCR R8,8 OUTPUT CONTROL DATA R10
0945 0370 1E0D SBZ 13 IGNORE INTERVAL DATA R02
0946 0372 04C8 CLR R8 R11
0947 0374 D224 MOVB @LPDSPX(R4),R8 GET TRANSMIT BAUD CODE R10
      0376 0076
0948 0378 0978 SRL R8,7 PUT CODE*2 IN LOW BYTE R10
0949 037A A248 A R8,R9 INDEX INTO BAUD RATE TABLE R10
0950 037C C259 MOV *R9,R9 GET CLOCK VALUE R10
0951 037E 3009 LDCR R9,0 SET TRANSMIT BAUD RATE R06

```

Figure 5-14. DSR Listing Example (Sheet 12 of 16)

```

0952 0380 1026      SBO INT902          ENABLE INTERRUPTS      R01
0953 0382 1027      SBO ABLTRS          ENABLE TRANSMITS      R01
0954 0384 1012      SBO RIENB           ENABLE READ INT FOR RO R01
0955 0386 1010      SBO RTS902          R01
0956 0388 1022      SBO SECRTS          SET SECONDARY RTS     R01
0957 038A 1020      SBO DTR902          SET DTR                R01
0958 038C 1004      JMP PWR2            R01
0959 038E 1009      PWR120 SBO EIADTR    INITIALIZE EIA INTERFACE R01
0960 0390 3020      LDCR @INIT,0
0961 0392 034E'
0961 0394 1E0F      SBZ EIADIM          FE
0965 0396 C186      PWR2  MOV R6,R6     THIS INITIAL POWER UP?
0966 0398 130C      JEQ NOTBZY
0967 039A 0262      ORI R2,>8000//DSFREN NO, SET RE-ENTER-ME
0968 039C 0400
0968 039E 0380      RTWP
0969
0985 03A0 C024      *
0985 03A2 0066      LPSPUR MOV @LPDIFF(R4),R0 DM INTERFACED LP ?
0989 03A4 1506      JGT NOTBZY          YES =
0990 03A6 1305      JEQ NOTBZY          YES =
1000 03A8 0A40      SLA R0,LPF902+1    9902 CONTROLLED ?    R01
1001 03AA 1702      JNC LPSP05          -NO.                  R01
1002 03AC 1E13      SBZ XBIENB          -YES. RESET INTERRUPT R01
1003 03AE 1001      JMP NOTBZY
1004 03B0 1E0B      LPSP05 SBZ EIAWRQ   EIA RESET INTERRUPT   R01
DSRLP SDSMAC 3.5.0 82.130 16:53:03 FRIDAY, JUN 17, 1983.
DSRLP - DNOS LINE PRINTER DSR PAGE 0021
1005 03B2 0206      NOTBZY LI R6,LPSPUR AND IGNORE IT
1006 03B4 03A0'
1006 03B6 0380      RTWP
1007 03B8 11      DC1 BYTE >11      READY SIGNAL
1008 03B9 13      DC3 BYTE >13      BUSY SIGNAL
1009 A200      CNTL25 EQU >A200 CONTROL REGISTER DATA FOR R06
1010
1011 *          9902
1012 *          1 STOP BIT
1013 *          EVEN PARITY
1014 *          CLK4 = 0 .. 2.5 MHz CLK
1015 AA00      CNTL4 EQU >AA00 7-BIT CHARACTER
1016 *          CONTROL REGSISER DATA FOR R06
1017 *          A 4MHZ 9902
1018 *          R06
1019 *          *****
1020 *          * THESE TABLES ARE USED TO CONVERT THE COMMON TRANSMIT R10
1021 *          * AND RECEIVE BAUD RATE CODE PASSED TO THIS ROUTINE R10
1022 *          * TO SECONDARY CODE FOR SETTING 9902 BAUD RATE. R10
1023 *          * R10
1024 03BA FFFF      BR25S2 DATA >FFFF 0 BAUD RATE 50 NOT SUPPORTED R10
1025 03BC 06B6      DATA >06B6 1 BAUD RATE 75 R10
1026 03BE 05D9      DATA >05D9 2 BAUD RATE 110 R10
1027 03C0 0583      DATA >0583 3 BAUD RATE 134.5 R10
1028 03C2 055B      DATA >055B 4 BAUD RATE 150 R10
1029 03C4 0504      DATA >0504 5 BAUD RATE 200 R10
1030 03C6 04AE      DATA >04AE 6 BAUD RATE 300 R10
1031 03C8 02B6      DATA >02B6 7 BAUD RATE 600 R10
1032 03CA 015B      DATA >015B 8 BAUD RATE 1200 R10
1033 03CC 00E7      DATA >00E7 9 BAUD RATE 1800 R10
1034 03CE 00AE      DATA >00AE A BAUD RATE 2400 R10
1035 03D0 0074      DATA >0074 B BAUD RATE 3600 R10
1036 03D2 0056      DATA >0056 C BAUD RATE 4800 R10
1037 03D4 003A      DATA >003A D BAUD RATE 7200 R10
1038 03D6 002B      DATA >002B E BAUD RATE 9600 R10
1039 03D8 001D      DATA >001D F BAUD RATE 14400 R10
1040 03DA FFFF      DATA >FFFF 10 BAUD RATE 19200 R10
1041 03DC FFFF      DATA >FFFF 11 BAUD RATE 28800 R10
1042 03DE FFFF      DATA >FFFF 12 BAUD RATE 38400 R10
1043 03E0' BR25E2 EQU $ MAX TABLE INDEX R10
DSRLP SDSMAC 3.5.0 82.130 16:53:03 FRIDAY, JUN 17, 1983.
DSRLP - DNOS LINE PRINTER DSR PAGE 0022
1045 *          R10
1046 *          9902 4.0 MHZ BAUD RATE TABLE R10

```

Figure 5-14. DSR Listing Example (Sheet 13 of 16)

Writing a DSR

```

1047 03E0 FFFF BR40S2 DATA >FFFF          0  BAUD RATE 50 NOT SUPPORTED R10
1048 03E2 0741 DATA >0741          1  BAUD RATE 75 R10
1049 03E4 0638 DATA >0638          2  BAUD RATE 110 R10
1050 03E6 05D1 DATA >05D1          3  BAUD RATE 134.5 R10
1051 03E8 05A1 DATA >05A1          4  BAUD RATE 150 R10
1052 03EA 0539 DATA >0539          5  BAUD RATE 200 R10
1053 03EC 04D0 DATA >04D0          6  BAUD RATE 300 R10
1054 03EE 0341 DATA >0341          7  BAUD RATE 600 R10
1055 03F0 01A1 DATA >01A1          8  BAUD RATE 1200 R10
1056 03F2 0116 DATA >0116          9  BAUD RATE 1800 R10
1057 03F4 00D0 DATA >00D0          A  BAUD RATE 2400 R10
1058 03F6 008B DATA >008B          B  BAUD RATE 3600 R10
1059 03F8 0068 DATA >0068          C  BAUD RATE 4800 R10
1060 03FA 0045 DATA >0045          D  BAUD RATE 7200 R10
1061 03FC 0034 DATA >0034          E  BAUD RATE 9600 R10
1062 03FE FFFF DATA >FFFF          F  BAUD RATE 14400 R10
1063 0400 001A DATA >001A         10  BAUD RATE 19200 R10
1064 0402 FFFF DATA >FFFF          11  BAUD RATE 28800 R10
1065 0404 000D DATA >000D          12  BAUD RATE 38400 R10
1066 0406' BR40E2 EQU $              MAX TABLE INDEX R10
1067 *
1075                                END R10
NO ERRORS, NO WARNINGS
DSRLP SDSMAC 3.5.0 82.130 16:53:03 FRIDAY, JUN 17, 1983.
LABEL VALUE DEFN REFERENCES PAGE 0023
$ 0406' 0338 0500 0649 1043 1066
ABLTRS 0027 0235 0953
ABORT 0326' 0890 0283 0284
ABRT10 0338' 0908 0891
BASE 0022' 0316 0338
BR25E2 03E0' 1043
BR25S2 03BA' 1024 0939
BR40E2 0406' 1066
BR40S2 03E0' 1047 0943
BRSTAT R 0020' 0120 0311
C$DNOS 0004 A0014 0004 0417
C$DPOS 0004 A0013 0028 0041 0138 0147 0250 0270 0494 0545 0568
0591 0612 0625 0681 0782 0840 0901 0919 0963
0987
C$DX10 0002 A0011 0009 0086 0109 0122 0160 0287 0411 0504 0550
0574 0596 0617 0632 0686 0759 0834 0894 0912
0972 0993 1069
C$OS 0004 A0019 0004 0009 0028 0041 0086 0109 0122 0138 0147
0160 0250 0270 0287 0411 0417 0494 0504 0545
0550 0568 0574 0591 0596 0612 0617 0625 0632
0681 0686 0759 0782 0834 0840 0894 0901 0912
0919 0963 0972 0987 0993 1069
CLOCK 0024 0237 0940
CLOSE 007A' 0384 0319
CNTL25 A200 1009 0938
CNTL4 AA00 1015 0942
CR 0076' 0381
CRLF 0078' 0382
DC1 0388' 1007 0872
DC3 0389' 1008 0866
DCD902 0020 0226
DFGPPF 0005 C0028 0907
DMCIN 000F 0220 0846 0926
DMDMD 000D 0218 0650
DMINT 000E 0219 0927
DMOUT 01D0' 0649 0628 0629 0679 0693
DMSTR 0007 0212 0671 0673 0929
DMSTRJ 0008 0213 0665 0667 0930
DMVFC 0009 0214 0928
DONE 0158' 0514 0498
DSCENB 0015 0242 0705 0852
DSFJAR 0006 C0045 0787 0789 0796 0798
DSFJIS 0004 C0043 0661 0756
DSFREN 0005 C0044 0967
DSR902 001B 0227 0702 0879
DTR902 0020 0240 0957
DUMP 013C' 0472 0337

```

Figure 5-14. DSR Listing Example (Sheet 14 of 16)

PDTWC	0046	C0076	0350	0351	0352							
DSRLP	SDSMAC 3.5.0		82.130	16:53:03		FRIDAY, JUN 17, 1983.						
LABEL	VALUE	DEFN	REFERENCES									PAGE 0025
PR9902	0003	0252										
PWR1	0350'	0933	0922									
PWR110	036E'	0944	0941									
PWR120	038E'	0959	0934									
PWR2	0396'	0965	0931	0958								
PWRON	033C'	0921	0282									
R0	0000		0627	0641	0653	0677	0700	0723	0748	0832	0849	
			0864	0865	0866	0872	0876	0877	0921	0933	0985	
			1000									
R1	0001		0373	0374	0375	0392	0393	0454	0454	0458	0463	
			0463	0467								
R10	000A		0310	0530	0531	0547	0565	0570	0571	0609	0614	
			0678	0683	0707	0710	0727	0729				
R11	000B		0461	0470	0496	0497	0707	0710	0727	0729	0805	
R12	000C		0400									
R2	0002		0661	0756	0787	0789	0796	0798	0967			
R4	0004		0405	0429	0434	0472	0496	0501	0530	0547	0558	
			0571	0584	0593	0603	0605	0608	0627	0674	0675	
			0677	0692	0714	0715	0736	0737	0792	0801	0831	
			0831	0832	0870	0875	0876	0887	0890	0903	0907	
			0921	0947	0985							
R5	0005		0307	0374	0375	0387	0405	0406	0419	0427	0432	
			0436	0439	0453	0459	0468	0563	0563	0585	0890	
			0908									
R6	0006		0388	0604	0623	0746	0847	0888	0909	0965	0965	
			1005									
R7	0007		0308	0373	0384	0385	0386	0392	0457	0466	0583	
R8	0008		0394	0396	0398	0400	0402	0404	0408	0424	0426	
			0429	0431	0434	0435	0438	0457	0466	0605	0608	
			0609	0614	0652	0674	0678	0683	0692	0714	0736	
			0747	0791	0800	0938	0942	0944	0946	0947	0948	
			0949									
R9	0009		0558	0565	0570	0583	0603	0652	0655	0657	0659	
			0660	0664	0670	0711	0713	0730	0731	0747	0750	
			0752	0754	0784	0784	0790	0799	0939	0943	0949	
			0950	0950	0951							
RCOPY	0126'	0463	0437									
RDST10	00C8'	0409	0407									
RDST20	00D2'	0425	0420									
READST	008A'	0392	0323									
RESET	001F	0241	0935									
REWIND	007E'	0386	0320	0322	0331	0332						
RFILL	0110'	0454	0428	0433	0440	0460	0469					
RFILL1	010C'	0453	0395	0397	0399	0401	0403	0425	0430			
RIENB	0012	0245	0855	0954								
RRQ902	0010	0229	0853									
RTS902	0010	0247	0536	0955								
SECDCD	0022	0225										
SECRTS	0022	0239	0956									
STAB	004E'	0339	0310									
TST010	0184'	0565	0586									
TST020	018A'	0571	0566									
TST030	01A0'	0603	0564	0582								
TST035	01B2'	0608	0606									
TST040	01BC'	0623	0610									
TST210	01EA'	0660	0654	0656	0658							
TST220	01FC'	0670	0663									
TST230	01FE'	0671										
TST240	0204'	0674	0668									
TST440	0266'	0735	0732									
DSRLP	SDSMAC 3.5.0		82.130	16:53:03		FRIDAY, JUN 17, 1983.						
LABEL	VALUE	DEFN	REFERENCES									PAGE 0026
TST450	0268'	0736	0734									
TST480	0270'	0738	0537	0541	0607	0651	0706	0716	0724	0726		
TSTDSR	0178'	0547	0389	0534	0623	0676						
TSTRR9	017C'	0558										
TSTW05	016E'	0539	0532									
TSTWR1	0176'	0542	0540									
TSTWRQ	015A'	0530	0746									
WDFFFF	0020+	J0024	0472									
WRITE	0062'	0373	0329	0330								
XBIENB	0013	0244	0535	0704	0708	0851	1002					
XBRE	0016	0228	0533									

Figure 5-14. DSR Listing Example (Sheet 16 of 16)

DNOS Accounting System

6.1 INTRODUCTION

At key points while executing each job in the system except the system job, DNOS collects information about utilization of resources. For instance, when each task terminates, an entry is logged in the accounting file. The entry identifies the task and job under which the task executed and includes central processing unit (CPU) utilization and memory allocation data.

The information is written in a compressed form to one of two accounting files, `.$ACT1` or `.$ACT2`. The system requires two files. One is being written and the other is available to be processed. The system switches to the other file when the file being written has been filled. If you examine these files using the Show File (SF) command, they appear as binary data. The files need to be processed by an application program to make use of the data.

The application program must process the entries in an accounting file before the system fills the other accounting file. When one accounting file becomes full, the system accounting routine starts to write the other file without determining whether or not the previous contents have been processed.

You must supply the application program to process the accounting file. When the system switches files, it bids the application program. This application program can process the data directly, copy the information to a magnetic tape, or possibly advise the operator that the file is full.

6.2 ACCOUNTING DATA

Accounting information for all jobs is written to the currently active accounting file in chronological order. The information for each job-related entry includes the job ID. The application program can sort the file on the job ID field to organize the information for each job.

The DNOS accounting file contains six types of entries. The application program must process all types of entries as appropriate for the accounting requirements of the site. There is also an identification record as record zero of the file; this record must be ignored by the application program.

6.2.1 Description of Accumulated Data

The six entries allowed in a DNOS accounting file are as follows:

- Job initialization — Contains the account ID, the job name, the job priority, and the user ID.
- Task termination — Contains the task ID, the CPU time used, the fixed priority (based on installed priority and job priority), the maximum amount of memory used, the termination code, and the number of supervisor calls (SVCs) issued.
- Job termination — Contains the amount of job communication area (JCA) memory used and the amount of JCA memory allocated when the job was created.
- Spooler device — Contains the job ID, the device name, the device type, and the number of I/O requests.
- User-defined — Supplied by the user task via an SVC. The string from the buffer for the SVC is the entry, along with additional information supplied by DNOS.
- Initial program load (IPL) — Contains the job ID. The IPL entry implies task termination for previously active tasks and job termination for previously active jobs.

Ten bytes of overhead are added to each entry. These bytes contain the type of entry, the number of bytes in the entry, the time when the entry was made, priority, and job identification.

6.2.2 Data Format

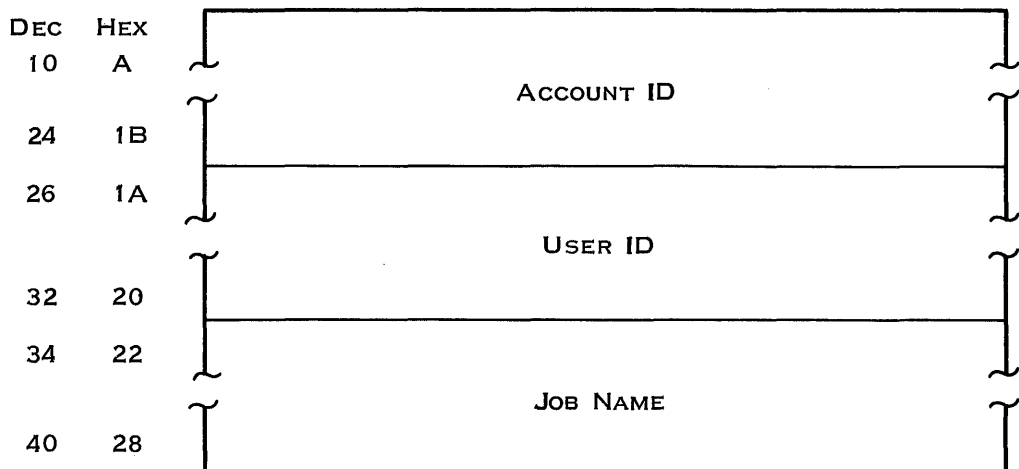
The first ten bytes of all accounting entries contain the following information:

DEC	HEX		
0	0	RECORD TYPE	RECORD LENGTH
2	2	YEAR/DAY	
4	4	HOUR	MINUTE
6	6	SECOND	PRIORITY
8	8	JOB ID	

2279424

Byte	Meaning
0	Record type. Value defines entry, as follows: 1 — Job initialization 2 — Task termination 3 — Job termination 4 — Spooler device 5 — User defined 6 — IPL
1	Record length, in bytes
2–3	The year and day. The two least significant digits of the year, in binary form, occupy the seven most significant bits. The day of the year (1 – 366) in binary form occupies the nine least significant bits.
4	Hour
5	Minute
6	Second
7	Priority. For a job initialization entry, the job priority. For a task termination entry, the initial priority of the task. Ignored for other entries.
8–9	Job ID

The additional bytes of a job initialization entry contain the following information:



2279425

Byte	Meaning
10 – 25	Account ID
26 – 33	User ID
34 – 41	Job name

For a task termination entry, the additional bytes contain the following information:

DEC	HEX	
10	A	TASK ID
12	C	TASK CPU TIME
14	E	
16	10	SVC COUNT
18	12	
20	14	I/O TRANSFER COUNT
22	16	
24	18	MAXIMUM MEMORY ALLOCATION

2279426

Byte	Meaning
10	Task run-time ID
11	Task termination code
12 – 15	Task CPU time (the count of clock cycles during task execution)
16 – 19	SVC count (the number of SVCs the task issues)
20 – 23	I/O transfer count (the number of bytes transferred during I/O)
24 – 25	Maximum memory allocation, in beets (1 beet = 32 bytes)
26 – 29	Elapsed time

Byte	Meaning
30	Installed task ID
31	Station number
32 – 33	Task segment attributes (attributes of segment as installed on program file)
34 – 41	Task name (as many as eight characters)

The additional bytes of a job termination entry contain the following information:

DEC	HEX	
10	A	JCA MEMORY USED
12	C	TOTAL JCA ALLOCATION

2279427

Byte	Meaning
10 – 11	JCA memory used (number of bytes of JCA used)
12 – 13	Total JCA allocation (number of bytes of JCA allocated for the job)

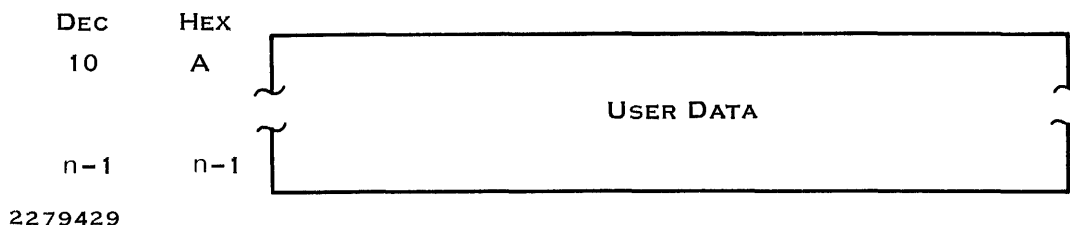
For a spooler device entry, the additional bytes contain the following information:

DEC	HEX	
10	A	DEVICE TYPE
12	C	DEVICE TYPE FLAGS
14	E	DEVICE NAME
16	10	NUMBER OF I/O REQUESTS
18	12	
20	14	TIME USED

2279428

Byte	Meaning
10	Device type flags
11	Device type
12 – 15	Device name
16 – 19	Number of I/O requests
20 – 21	Time used (reserved for a count of minutes of use)

For a user device entry, the additional bytes contain the following user data:



Byte	Meaning
10 – n	User data supplied by the Log Accounting Entry SVC (> 47). This is the entire contents of the buffer defined in the SVC block, less the first byte which contains the byte count. The maximum number of bytes is 70.

The IPL entry includes only the data in the initial ten bytes.

6.3 IMPLEMENTATION

To implement the accounting subsystem, you must do the following:

- Define account numbers.
- Generate a system that includes the accounting subsystem.
- Provide a task to process the accounting information.

6.3.1 Account Numbers

Account numbers are user-defined character strings 16 bytes in length. DNOS does not impose any other restrictions.

Account verification is optional and requires a file of valid account numbers. The name of this file must be `.$ACCVAl`. This file must be a sequential file with one account number per record. Be sure to include a blank record in the file if any users can log on without an account number. Without this blank record, all jobs in the system must use an account number listed in the file. Write the file of valid account numbers to provide account verification. When file `.$ACCVAl` has not been created, account numbers are not validated.

6.3.2 System Generation Requirements

The accounting subsystem is an option that can be included when a DNOS system is generated. A group of SVCs (the accounting group) consists of the two SVCs required to support job accounting. By including the accounting group of SVCs, you implicitly request that the accounting subsystem be included. Nothing else is required to generate a system that supports job accounting.

6.3.3 Application Program Requirements

The third requirement for job accounting is an application program to retrieve and process the information on the accounting file. This task must be installed on the system utilities program file `.$UTIL` as task ID > 54, which is reserved for the user accounting task. When an accounting file is full, DNOS starts the task installed at ID > 54.

The application program can determine the file that needs processing by issuing a Get Task Parameters SVC. The third byte of the task parameters contains a 1 in ASCII form if the first file needs processing and a 2 in ASCII form if the second file needs processing. If the processing task is written in Pascal, it must be linked with the MINOBJ routine so that stack and heap parameters are not expected as task bid parameters.

Processing the entries on one file while the other file is being written, the application program can copy the entries to a more permanent file to be processed offline or at a later time.

File Security

7.1 INTRODUCTION

In a DNOS system using file security, a user can perform an operation on a file only if the following two conditions are met:

- You are a member of an access group that has access rights to the file.
- The operation is allowed by the access rights that the access group has to the file.

You should refer to the *DNOS System Command Interpreter (SCI) Reference Manual* for explanations concerning any commands with which you are not familiar. Read this entire section before you attempt to use file security. If you have any questions concerning file security, talk to the security manager of your system.

7.2 ACCESS GROUPS

An access group is composed of a set of user IDs. The users of these IDs usually have a common work assignment or a common need for system resources.

The name of an access group must be a string of one to eight alphanumeric characters, with the first character alphabetic. Access rights to particular files on the system are assigned by access group name. Each secured file on the system can have access rights defined for up to nine access groups. You can define a different set of rights for each group.

There are two types of access groups: pre-defined and user-defined. PUBLIC and SYSMGR are the only two predefined access groups.

Everyone on the system is automatically an access group member of PUBLIC. A user who belongs only to the PUBLIC access group has rights only to unsecured files and files that creators have defined for PUBLIC access.

Usually, only the security manager or a small, trusted group belongs to the SYSMGR access group. The SYSMGR group has access rights to every file and implied leadership of every access group.

All other access groups on the system are user-defined and are created for a specific purpose. The creator of an access group automatically becomes the initial access group leader. The leader designates which users on the system are members of his access group. A user can be a leader of one or more access groups and also a member in others.

After creating a file, a user can specify which access groups are allowed access to it and the types of access these groups can have.

The roles one can play in access groups are briefly described in the following paragraphs.

7.2.1 SYSMGR Access Group Member

Because a member of the SYSMGR access group can access every file in the system and assume leadership for every access group, this group is not to be used for routine tasks. The SYSMGR access group should be limited to three special functions:

- Setting up the security environment
- Solving of unusual system problems
- Applying patches supplied by Texas Instruments

The security manager is normally either the only member of SYSMGR or the leader of a small, trusted group.

7.2.2 Access Group Leader

A user on the system becomes an access group leader either by creating an access group or by having the leader of an access group give up his leadership and designate him as the new leader.

Each access group has only one leader (however, any member of the SYSMGR access group can perform any function the leader can). The access group leader controls membership in the group by his right to add or delete members. To create an access group, a user must have access to the Create Access Group (CAG) command procedure. User IDs associated with the SYSMGR access group cannot be used with the CAG command.

7.2.3 Access Group Member

An access group member is a person whose user ID belongs to the set of user IDs for a specific access group. Only the access group leader (and members of the SYSMGR access group) can add or delete access group members. An access group member shares the access rights of his group to files on the system. Every user is a member of at least one access group since all users of the system belong to PUBLIC.

7.2.4 Creation Access Group

A user's creation access group is the access group that has all access rights to files he creates. Every user on the system has exactly one creation access group. A user must select one of the access groups to which he belongs as his creation access group. A user makes this selection by executing the Set Creation Access Group (SCAG) command. If a user never specifies a particular group, his creation access group is PUBLIC by default.

As long as the creation access group retains control access, only users who belong to the group (or members of SYSMGR) can give any access rights to a file to other access groups (by executing the Modify Security Access Rights (MSAR) command). Giving read, write, execute, or delete access rights to another group does not take away these rights from a user's own group. However, since only one access group can have control access to a file, giving control access to another access group means taking it away from the user's group. Therefore, a user should be very careful before assigning control access for a file to another access group.

For example, a user selects an access group called LAWYERS as his creation access group. He logs off and logs on again (the system recognizes the selection only in this manner). Then, he creates a file called ACCOUNTS. The LAWYERS access group now has control, read, write, execute, and delete access to ACCOUNTS. Later, he decides that a related access group, called PARALGLS should have read access and write access to the file. He would then use the MSAR command to assign them these two rights. Later, unknown to him, a junior member of LAWYERS uses the MSAR command to assign control access to the PARALGLS access group. The next time he tries to assign access rights to ACCOUNTS with the MSAR command, he will receive an error, since his access group, LAWYERS, no longer has control access to the file. After locating and taking appropriate revenge on the junior member of LAWYERS who gave the file away, he would have to talk one of the members of PARALGLS (or a member of SYSMGR) into using the MSAR command to return control access to LAWYERS.

7.2.5 Modifications to Access Groups and Access Rights

The system determines what access groups a user belongs to when a job is created with that user's ID. Therefore, for the system to recognize any selection or modification that involves access groups, the user affected by such a change must log off and then log back on again for the change to take place. Changes to membership in an access group or to the selection of current creation access group do not affect running jobs.

Modifications to the access rights of a file take effect immediately. However, the system determines a user's access rights to a file when he tries to assign a LUNO to the file. Therefore, if a LUNO is already assigned to a file in a job, the job will execute regardless of any changes to the file's security.

For example, you are running a job that uses a file to which your access has all access rights. If someone uses the Modify Security Access Rights (MSAR) command to take away all the access rights to the file, the job you are running may or may not be affected. If a LUNO is already assigned to the file, the modification will not affect the running job. However, if the LUNO has not been assigned, an error occurs when the task (under which the job is running) tries to access the file for which you no longer have access rights.

7.3 ACCESS RIGHTS

There are five possible types of access rights to a file:

- Control access
- Read access
- Write access
- Execute access
- Delete access

The access rights that an access group possesses define the ways in which its members can use a particular file.

Only one access group can have control access to a file. However, a member of an access group that has control access to a particular file can give any access group any combination of the first four rights to the file.

You can use the MSAR command to assign or alter access rights to a file. When securing a file, you should carefully consider security requirements before deciding which groups should have access rights to the file. You should consider whether or not certain access rights can be withheld without affecting the normal work of the users. The following paragraphs describe each of the access rights.

7.3.1 Control Access

Control access is the right to change the access groups associated with a file or to change the access rights of any access group. Only one access group can have control access to a particular file. You must be part of an access group that has control access to a file in order to execute the MSAR command on that file.

7.3.2 Read Access

Read access is the right to read the contents of a file. This right also enables you to execute a file if it is an SCI batch stream or command procedure. If the file is a program file, read access allows you to designate the file when issuing the Map Program File (MPF) and Show Program Image (SPI) commands. With read access to a file, you can copy the contents of the file into any file for which you have write access.

7.3.3 Write Access

Write access is the ability to write data into a file. It allows you to modify old data and write new data. In addition, write access to a program file enables you to install or delete tasks, segments, procedures, and overlays. If the file is a key indexed file, this right allows you to insert or delete records from the file.

7.3.4 Execute Access

Execute access applies only to program files. This right allows you to execute tasks, segments, procedures, and overlays within a program file. As a security measure, you can protect your powerful or sensitive tasks by placing them in a protected program file. However, do not move tasks supplied by Texas Instruments, as this step would affect the ability of the system to accept Texas Instruments patches.

7.3.5 Delete Access

Delete access is the right to delete (or replace) a file. In order to text edit a file, you must have both write and delete access.

7.4 EXAMPLE OF A SECURED SYSTEM

Figure 7-1 shows the relationship between access groups and access rights to particular files. Imagine a system that currently has only two user-defined access groups (ADMIN and FINANCE) and two secured files (ACCOUNTS, and BILLING). PRES is the access group leader for ADMIN. He has designated ADMIN as his creation access group.

ACCESS GROUP	IDS OF MEMBERS	
ADMIN	PRES , VP	
FINANCE	CMPTRLR , ANALYST , CLERK1 , CLERK2	

SECURED FILE	ACCESS GROUP	ACCESS RIGHTS
ACCOUNTS	ADMIN	READ , DELETE , CONTROL
	FINANCE	READ , WRITE
BILLING	ADMIN	READ , WRITE , DELETE , CONTROL
	FINANCE	READ

2284929

Figure 7-1. Access Groups and Secured Files

In this system, PRES can read both of the secured files because read access to these files has been defined for the ADMIN access group. He can also write to the BILLING file. ANALYST can write to ACCOUNTS because write access to the file has been defined for the FINANCE access group. However, if he tries to write to the BILLING file, he will receive an error because write access has not been defined for his access group.

Consider that PRES needs to modify the accounts that his company has. To do so, he must establish write access to the ACCOUNTS file. Because he belongs to the ADMIN access group (which has control access to that file), he may issue the MSAR command to give write access to ADMIN.

Consider that ANALYST was convinced he could help ADMIN modify the BILLING file. He has two options. First, he could ask PRES (the access group leader) to include his user ID in the ADMIN access group. Second, he could ask anyone in the ADMIN access group to modify the security of the file in order to give write access to his access group, FINANCE.

Consider that PRES wants to create a stock strategy file which only the PRES, VP, and CMPTRLR can access. He can create an access group called LEADERS by executing the CAG command. PRES automatically becomes access group leader of LEADERS because he executed the command. If he wants to designate someone else as the leader he must execute the Modify Access Group (MAG) command. Then, he can use the Create File (CF) command to create a file named STRATEGY. The ADMIN group now has all access rights to the file (because PRES has selected ADMIN as his creation access group). He can then execute the MSAR command to define what access rights he wants the LEADERS access group to have.

After these operations, the system would have one additional access group and one additional secured file, which might appear as follows:

ACCESS GROUP	IDS OF MEMBERS	
LEADERS	PRES , VP , CMPTRLR	

SECURED FILE	ACCESS GROUP	ACCESS RIGHTS
STRATEGY	ADMIN	READ , WRITE , DELETE , CONTROL
	LEADERS	READ , WRITE

2284930

Figure 7-2. Creating an Access Group

7.5 IMPORTANT POINTS ABOUT ACCESS RIGHTS TO SECURED FILES

Each secured file can have access rights defined for up to nine access groups. The MSAR command assigns access rights and modifies them. You can define a different set of rights for each group.

The rights granted to an access group define the rights of each member. The rights of an individual user are determined by membership in access groups. A user can access a file only if he is part of an access group that has rights to the file.

The rights to a file for a given user are a composite of the rights for all of the access groups to which he belongs. For example, a user is a member of two access groups. The first has read access to a file and the second has write access to the same file. In this case, the user has both read and write access to that file.

Access rights are independent of each other. Any combination of rights can be assigned to a file, even if certain combinations would not appear to make much logical sense.

The system establishes what access groups a user belongs to when a job is created with that user's ID.

The system establishes a user's access rights to a file when he assigns a LUNO to the file.

The write-protect, delete-protect mechanisms of the Modify File Protection (MFP) are independent of file security, except in one way. To use the MFP command on a file, you must have write access and delete access.

Changing the data in a file or the name of a file does not affect the access rights associated with the file. However, if you delete a file and create a new one with the same name, the file is no different from any other that you create under your user ID: your creation access group inherits all access rights to the new file.

Programs that copy files normally read the data from an input file and write the data to an output file. If the copying process is executed with any of the following SCI commands, the security rights of the input file are transferred to the output file:

- BDD—Back Up Directory to Device (followed by the Restore Directory (RD) command)
- CV—Copy Volume
- CVD—Copy Volume to Device
- DCOPY—Disk Copy/Restore

If the copying process is executed with any other command, the security of the input file is not transferred to the output file.

7.6 PROGRAMMERS

The following facilities related to security are available to programmers:

- I/O utility operations that specify a user ID
- Tasks designated as security bypass tasks
- Special Rename File SVC option
- Open routine specifying user ID (\$OPNS)
- No echo option for SCI prompt response
- Read file characteristics security option

7.6.1 I/O Utility Operations That Specify a User ID

In most cases, when a task uses a file, it does so with the access rights of the user ID of the job in which it is running. In other cases, the task may be a special request server that runs in its own job. In the latter case, the task may need to access a file with the access rights of the requesting task. The user ID and passcode of the requesting task are specified as an I/O parameter in the Supervisor Call (SVC) block for I/O utility operations in the request server task. For security bypass tasks, the passcode does not need to be specified. A security bypass task that specifies a user ID in the parameter list does not bypass security checking for the specified I/O operation. Instead, it picks up the access rights associated with the specified user ID. To set up an SVC block that specifies a user ID, refer to the *DNOS Supervisor Call (SVC) Reference Manual*.

The following I/O utility operations may specify a user ID as an SVC block parameter:

- Assign LUNO—This operation assigns a LUNO if the specified user has any access rights to the file. All subsequent I/O operations that use the LUNO are verified against the specified user's access rights.
- Create File—This operation creates a file with full access rights given to the creation access group of the specified user.
- Delete File—This operation deletes a file if the specified user ID has delete access to the file.
- Unprotect File—This operation 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 operation write protects a file if the specified user ID has write and delete access to the file.
- Delete Protect File—This operation delete protects a file if the specified user ID has write and delete access to the file.

7.6.2 Security Bypass

Security bypass gives access rights to a program without giving it to the user. The Modify Task Security Attribute (MTSA) command assigns security bypass to a task in a program file; it can also remove this privilege.

A task that is installed with the security bypass attribute is granted access to any file on the system (except any task that uses an I/O utility operation specifying user ID). For this reason, you should secure the MTSA command under your security manager maintenance access group, so no one but you can assign the security bypass attribute to a task. It is your responsibility to guarantee the integrity of the task, as the task itself must enforce security. You, or a trusted programmer, should look over the logic of the task to insure that no unnecessary files are accessed. Once you have approved the task, you should perform the following steps:

1. Assign the security bypass attribute to the task with the MTSA command.
2. Use the MSAR command to give the control, delete, and write access rights for the program file to your security manager maintenance group. With read and execute access, the user can use the program file for the needed purpose but he does not have the ability to modify it.
3. Write protect the program file for the task, so the file cannot be modified.

A security bypass task that uses one of the I/O utility operations that specify user ID is affected as follows:

- The task inherits the access rights of the user ID specified rather than the full access rights normally given to a security bypass task.
- The task does not need to specify the user passcode in the SVC parameter list.

To set up an SVC block for I/O utility operations that specify user ID, refer to the *DNOS Supervisor Call (SVC) Reference Manual*. Having a security bypass task use one of these operations is useful in cases where you want to give a task unlimited access to files during execution but you want normal file access security for input and output files.

For example, you are willing to give the security bypass attribute to a user's task but you want to guarantee that he can only place the output from the task into files for which the user has access rights. You can limit the user in this way by having the task use an Assign LUNO SVC that specifies his user ID when the task attempts to place the output in a specified file. At this point, the task loses its security bypass attribute. Therefore, before assigning the LUNO, the system checks whether the user has the proper access rights to the output file.

7.6.3 Special Rename File SVC Option

With the normal execution of the Rename File SVC, the new file assumes the security of the old file. For example, if you are modifying a file called LIST1 to be called LIST2, the LIST2 file assumes the security that belonged to LIST1.

However, the special Rename File option allows you to keep the security of the destination file (if it exists) rather than that of the source file. Refer to the *DNOS Supervisor Call (SVC) Reference Manual* for details about this option.

You can use the option under the following three conditions:

- The destination file already exists.
- The replace option is specified.
- You have write access to both the source file and the destination file.

7.6.4 Open Routine Specifying User ID (S\$OPNS)

The S\$OPNS routine performs the following functions:

- Executes an Assign LUNO SVC, specifying a user ID.
- Opens a user-specified file, a user-specified device, or the Terminal Local File (TLF) for write access. To perform the S\$OPNS routine, refer to the *DNOS Systems Programmer's Guide*.

A programmer should use this routine instead of the standard S\$OPEN routine when the following conditions are true:

- The calling task is a security bypass task.
- A standard security check on the listing file is desired. Therefore, a user who executes the task cannot place the output in a file for which he does not have access.

7.6.5 No-Echo Option for SCI Prompt Response

When writing SCI prompts, a programmer can use the no-echo option to indicate that data entered into a field is not to be displayed. Refer to the *DNOS System Command Interpreter (SCI) Reference Manual* for details.

7.6.6 Read File Characteristics Option

The Read File Characteristics operation of the I/O SVC has an option that allows the issuer of the SVC to determine what rights he has to a file. If the option is specified, the SVC returns a word of data in the specified buffer. The data indicates what access rights the issuer of the SVC has to the file.

The issuer of the SVC can also determine what access rights another user has to a file. If the user has access rights, he has previously been assigned a LUNO that specified his user ID. The issuer of the SVC can use this LUNO to find out what access rights the user has.

Refer to the *DNOS Supervisor Call (SVC) Reference Manual* for details about this option.

Analyzing System Problems

8.1 SYSTEM INITIALIZATION PROBLEMS

During the initial program load (IPL) process, the system crashes when a loader error is encountered. The three loaders used to load the DNOS operating system into memory are the read-only memory (ROM) loader, the program image loader, and the system loader. The following paragraphs discuss the error indications provided by the three loaders.

8.1.1 ROM Loader Errors

When an error occurs during ROM loader execution (TILINE load), the system crashes. The most common errors are controller and unit select errors. Regardless of the type of error, the fault light flashes on and off to indicate an error occurred. Depending on the error, the status of TILINE peripheral control space is displayed on the indicators of the programmer panel.

If the error is a controller error and you are using a 990/10 or 990/12 computer, one of the following indicators lights up on the front panel:

0	6	7	8	9	10	11	12	13	14	15
ALWAYS ZEROS	AC	ME	DE	TT	IE	RE	CT	SE	0	

2279430

AC — Abnormal completion	IE — ID error
ME — Memory error	RE — Rate error
DE — Data error	CT — Command timer
TT — TILINE time-out	SE — Seek error

Refer to the appropriate disk installation and operation manual for further information regarding error conditions.

If the error is a unit error and you are using a 990/10 or 990/12 computer, the following status word appears on the indicators:

0	1	2	3	4	5	6	15
OL	NR	WP	US	0	SI	UNKNOWN	

2279431

OL — Offline	US — Unsafe
NR — Not ready	SI — Seek incomplete
WP — Write protect	

8.1.2 Program Image Loader Errors

When the program image loader detects an error, it terminates with the flash crash routine. This routine displays >FF in the leftmost indicators, flashing on and off. The rightmost indicators show either of the following error codes:

- >01 — Error loading from the disk
- >08 — Unable to locate loader file

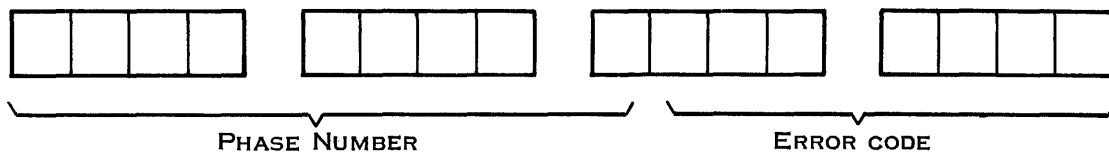
If the >01 error is displayed, disk track 1 information may have been destroyed. If the >08 error is displayed, either an invalid system loader file name has been specified in the volume information or the disk image of the system loader file is damaged.

8.1.3 System Loader Errors

When the DNOS system loader detects an error during the IPL, the way it reports the error to you depends on what kind of computer you are using. On a 990/10 or 990/12 system, the loader displays the error code in the rightmost seven indicators and the phase number in the leftmost nine indicators on the programmer panel. The phase number indicates the last successful phase executed, indicating that the loader detected an error in the next phase. The leftmost nine indicators flash on and off to indicate that the error occurred in the system loader. On a Business System computer, the hexadecimal equivalent of the error code is displayed in four digits on the front panel.

Figure 8-1 shows the programmer panel phase number and error code display on a 990/10 or 990/12 computer. The phase number is displayed as a bar graph starting from the leftmost indicator. Indicator 0 is lit at the completion of phase 1, indicator 1 is lit at completion of phase 2, and so on. Each indicator remains lit until the loader completes execution.

On a Business System Computer, the hexadecimal equivalent of the error code is displayed in four digits on the front panel. Table 8-1 lists the phase numbers. The error code is displayed as a binary number. Table 8-2 lists the codes and their meanings.



2279432

Figure 8-1. System Loader Error

Table 8-1. System Loader Phases

Indicator	Phase	Description
0	1	Successful loader relocation
1	2	Successful open of kernel program file
2	3	Successful load of writable control store (WCS), load of system root, and verification of system version
3	4	Successful loading of special table areas
4	5	Successful initialization of system overlay table and crash file
5	6	Successful loading of JCA segments
6	7	Successful loading of DSRs and scheduler
7	8	Successful loading of memory-resident system tasks
8	9	Successful loading of user memory-resident tasks

Table 8-2. System Loader (Flashing) Crash Codes

Error Code	Description
> 01	Load device I/O error
> 02	Not enough memory to load system
> 03	System disk not found in image file
> 04	Error in program file directory
> 05	Loader incompatible with system version being loaded
> 06	Disk bit map error
> 08	No system loader file
> 09	No kernel program file
> 0A	System segment not found in program file
> 0B	No patches applied to system
> 0C	Software version too old
> 0D	No utility program file
> 0E	No swap file
> 0F	Kernel program file revision is inconsistent with file revision
> 11	Unable to get system table area
> 13	Logical address space overflow
> 14	Unable to load WCS file
> 60-> 6F	Error interrupt (Level 2)
> 68	Insufficient user task area

Refer to the *DNOS Messages and Codes Reference Manual* for more diagnostic information on loader flash crashes.

8.2 SYSTEM CRASH PROBLEMS

When DNOS detects a system failure, it displays an error code on the front panel indicators and places the CPU in the idle mode. The fault indicator also lights to indicate a system crash. At this point, all the terminals stop responding to users.

To analyze the system crash, perform the following steps:

1. Press HALT and then RUN on the front panel to copy the contents of memory to the predefined crash file on disk.
2. Perform the IPL to load the system again.
3. Bid SCI at a terminal.
4. Enter the XANAL command to execute the crash analyzer.

To perform the IPL, press HALT and LOAD on the front panel. Refer to the *DNOS Operations Guide* for the complete procedures.

The Execute Crash Analysis Utility (XANAL) SCI command provides a formatted listing of the system crash file. A systems programmer can use this listing to analyze the cause of the system crash.

The commands given to XANAL select portions of the memory dump on the crash file (or actual memory if the running system is being analyzed) and write them to the listing device in a formatted form. Table 8-3 summarizes the XANAL commands.

Table 8-3. XANAL Commands

Command	Action
ALL	Execute all of the commands
AQ	Display contents of active queue
CCB	Channel control block
DM	Dump a specific area of memory
FCB	Dump file control blocks
GI	Display general information
JSB	Dump job status blocks (JSBs)
LDT	Logical device tables
MM	Memory maps
OVV	Overhead beets
PBM	Partial bit maps
PDT	Dump physical device tables
PQ	Display all other system queues
QU	Terminate session
ROB	Resource ownership blocks
RPB	Resource privilege blocks
SGB	Dump segment group blocks
SSB	Dump segment status blocks
ST	Dump secondary table areas
TA	Dump task areas currently in memory
TR	Dump registers for all tasks
TS	Display task status
TSB	Dump task status blocks (TSBs)
??	List all commands

Normally, you should execute the GI, TS, and JSB commands in sequence to obtain the general information about a crash. The programmer can then print some of the data structures (such as TSBs) and memory maps as needed. The data structures printed by XANAL are described in detail in the *DNOS System Design Document*.

Sometimes a great deal of knowledge about the system and system data structures is required to determine the underlying causes of the system crash from the XANAL listing. However, a systems programmer should be able to get much general information from the crash dump without getting involved in the details of the system structures.

If you cannot resolve a system crash, do the following:

- Report the system crash to a customer representative.
- Provide a copy of the crash file (.S\$CRASH) on a magnetic medium to the customer representative. (The medium will be returned after the data has been copied from it.)
- Also provide the three link maps of the system and a file describing the events that led to the crash.

8.2.1 Organization of the XANAL Listing

The first page of the crash dump generated by the GI command contains the general information block. Figure 8-2 shows an example of general information printed by the XANAL utility. The following paragraphs briefly describe each entry of the XANAL listing shown in Figure 8-2.

8.2.1.1 Crash Code. The first entry is the system crash code displayed on the front panel when the crash occurred and the meaning of that code. An example of the crash code display is as follows:

```
CRASH CODE = 00A2      FILMGR -- INCONSISTENT STRUCTURE
```

In this example, >A2 (leading zeros are omitted in this section) was the error code returned when file management detected an inconsistent data structure.

If the crash code is in the range of >60 through >6F (referred to as >6x crash), the crash code represents a level 2 interrupt error. An example of the crash code for a >6x crash is as follows:

```
CRASH CODE = 0062      ERROR IN OS - ILLEGAL INSTRUCTION
      /      \
     /        \
    /          \
   /            \
  /              \
 /                \
/                  \
CRASH TYPE      ERROR CODE
```

The third digit identifies a >6x crash and the last digit describes the cause of the crash.

8.2.1.2 Executing Task. The second entry is the address of the TSB of the task that was executing when the crash occurred. When this value is 0, no task was executing at the time of the crash. Therefore, the crash occurred within the operating system, probably during a scheduling cycle.

8.2.1.3 Executing Task JSB. The third entry is the address of the job JSB of the task that was executing when the crash occurred. The JSB contains the job name, job ID, and user ID of the job.

8.2.1.4 Location of Failure. This entry is the address from which the crash routine was called. In some cases, this entry points to the exact location of the crash. However, in most cases, this value is the location of a common crash point that can be entered from any of several locations.

8.2.1.5 Status Register. This entry lists the value of the status register when the crash occurred. The last four bits (last digit in the entry) of the status register are the interrupt mask. The status register information is valuable when the crash code is in the range of >10 through >1F, indicating an illegal interrupt. When an illegal interrupt occurs, the interrupt mask value is in the range of >2 through >E (refer to the paragraph on forcing a system crash). When the crash code is >6x, the interrupt mask value is 1. This value indicates that the crash was caused by one of the task error states occurring within the system or within a system task.

8.2.1.6 CURMAP Addr. This entry lists the value of the current map file at the time of the crash. This information is useful when the crash code is >61 (memory parity error). You can calculate the physical memory location using this information if the failure is in map 0. If the failure is in map 1, use the values of the map registers in the TSB.

SYSTEM DUMP ON 8/15/81 AT 10:17 -- VERSION 1.2.00

CRASH CODE = 0062 ERROR IN OS - ILLEGAL INSTRUCTION
 EXECUTING TASK = 0000
 EXECUTING TASK JSB = 0000
 LOCATION OF FAILURE = 0E92
 STATUS REGISTER AT TIME OF FAILURE = C401
 JCASTR=9000
 COUNTRY CODE =UNITED STATES
 IMAGE NAME= S*SHIP
 MEMORY SIZE 756K BYTES
 CRASH FILE SIZE 756K BYTES
 CURMAP ADDR = 4622

EXECUTING WORKSPACE AT TIME OF DUMP

30C6-0009 0000 FFFF 4622 000F 000F 0000 0000 F"
 30D6-0006 FE01 7F80 0000 1FE0 31A4 C190 301F 1. .. 0.

TOP 64 WORDS OF CURRENT STACK

7F40-0008 A356 4CFC A356 0000 96EE 0300 4CDA .. .V L. .V L.
 7F50-9D98 2D1E 0000 0180 0000 006E 4078 FOOD .. -. e. ..
 7F60-C316 8000 C354 70D2 30B3 0064 C354 CCAAT .. 0. .. .T ..
 7F70-0000 CD08 0024 A386 0000 A3E2 E5CE 0000 \$
 7F80-EA6C 0000 B050 4000 A356 0000 0000 CBEBP e. .V
 7F90-373A AFB6 B0B6 37BC 0000 70A0 7FB6 7F8E 7: 7.
 7FA0-0B00 7F44 0000 0300 4744 0300 3B4C 0000 .. .D GD .. 8L ..

 7FB0-37FE 0022 72AB 0000 5CC1 0000 5CC1 FOOD 7. ." \. .. \. ..

MACHINE ERROR (TRAP 2) WORKSPACE

0E7C-0000 0002 0000 0080 0000 0060 1FA0 1FC0
 0E8C-000F 0420 08A0 0062 1FC0 30C6 076C D004 0.

TRAPPED WORKSPACE

30C6-0009 0000 FFFF 4622 000F 000F 0000 0000 F"
 30D6-0006 FE01 7F80 0000 1FE0 31A4 C190 301F 1. .. 0.

LOCATIONS AROUND TRAPPED PC

075C-1503 0420 08A0 002C 2EC2 C0E0 2DD0 0000 .. .
 076C-2DCE C80B 2DD0 032B 0460 EBBC C803 2DD0 -. .. -. .+ -. ..

SVC (XQP 15) WORKSPACE

31A4-0000 0000 2E1A 0000 00F4 0000 0000 0000
 31B4-6FDC 91F4 31C4 C10E 0000 90A4 C08A 209F 1.

CLOCK INTERRUPT WORKSPACE

30C6-0009 0000 FFFF 4622 000F 000F 0000 0000 F"
 30D6-0006 FE01 7F80 0000 1FE0 31A4 C190 301F 1. .. 0.

HARDWARE TRAP VECTORS

0000-30F0 114A 30EA 113E 0E7C 0DD6 00C0 34FC 0. .J 0. .> 4.
 0010-00C0 34FC 30C6 0766 00C0 34FC 4BD6 3468 .. 4. 0. 4. K. 4.
 0020-00C0 34FC 4A44 34BE 458A 34BE 496A 34E2 .. 4. JD 4. E. 4. I. 4.
 0030-4926 34E2 4BE2 34E2 00C0 34FC 00C0 34FC I& 4. H. 4. .. 4. .. 4.

XQP VECTORS

0040-00C0 C606 6B18 6B46 00C0 C606 00C0 C606 F
 0050-00C0 C606 00C0 C606 00C0 C606 00C0 C606
 0060-00C0 C606 0000 0000 1C62 1CB2 1C2E 1C4E
 0070-1BA4 1BC4 00C0 C606 00C0 C606 31A4 C45C 1. .\

SYSTEM PATCH AREA

3CF0-028C 0256 2043 2043 4F50 5952 4947 4854 .. .V C C DP YR IG HT
 3D0C-2031 3938 3220 5445 5841 5320 494E 5354 1 98 2 TE XA S IN ST
 3D1C-5255 4D45 4E54 5320 494E 434F 5250 4F52 RU ME NT S IN CD RP OR
 3D2C-4154 4544 0010 0002 04E0 2DB0 C060 2DBC AT ED
 3D3C-0460 1326 0016 0000 07C3 C820 C626 000A .. .&& ..
 3D4C-07C3 C80F 0002 0460 D2C0 DEAD DEAD DEAD
 3F7C*DEAD DEAD DEAD DEAD DEAD DEAD

Figure 8-2. General Information Block

8.2.1.7 System Patch Area. This entry lists patches that have been applied to the system. Use this information to verify that all out-of-line patches have been applied to the system successfully. The beginning address of the patch area should be equal to the value assigned to NFPATCH. Examine the SYSMAP listing for your system to see the expected address. The last few lines of the patch area show the revision level of the release. Check the revision level to make sure that all recent patches have been applied. The patches are applied at the end of system generation.

8.2.1.8 Executing Workspace at Time of Dump. The executing workspace contains the registers of the executing code at the time of the crash.

8.2.1.9 Hardware Trap and Extended Operation (XOP) Vectors. Examine the transfer vectors for hardware interrupts and XOPs to verify that they are intact. Since these values reside in the first 64 words of physical memory, they can be destroyed by a system task that branches to memory location 0. Usually, the locations that are destroyed are locations > 0 through > 3 (power-up interrupt) or locations > 1A through > 1F. Locations > 1A through > 1F are destroyed when a task executes a BLWP instruction to location 0. When the BLWP instruction is executed, the return context information of the calling task is stored in locations > 1A through > 1F. When a > 6x crash occurs and the interrupt mask in register 15 of the workspace for interrupt level 2 indicates a defined interrupt (mask value minus 1), check the interrupt trap values to determine if they are within the proper address range. Except for interrupt levels 0, 1, 2, and 5, the workspace pointer and program counter for each interrupt level should contain addresses that are relatively close to each other.

8.2.1.10 Special Workspaces. The workspaces for the clock processor, the level 2 interrupt processor, and the SVC processor are printed last. These routines are entered through context switches and the return context is found in registers R13, R14, and R15 of these workspaces.

If the crash code is > 6x, XANAL prints out the active workspace at the time of the crash and 16 words of locations around the address in the program counter at the time of the interrupt.

The clock workspace contains the location in the executing task where the last clock interrupt occurred. The SVC workspace contains the location of the last SVC or the scheduling location. These two locations can sometimes help to determine where a task was executing at the time of the crash. The interrupt 2 workspace contains diagnostic information about a > 6x crash. R13 through R15 contain the context of the crash within the system.

When looking at the saved status register for interrupt 2 (R15), notice the value of bit 8. If bit 8 is set to 1, the crash occurred in task code (map 1). If bit 8 is set to 0, the crash occurred in system code (map 0).

8.2.2 Task States and JSBs

The TS and JSB commands list the task states and JSBs of all tasks in memory at the time of the crash. Figure 8-3 shows example lists of task states and JSBs.

TASK STATES

TASK NAME	ID	WP	PC	ST	STATE	FLAGS	STATION	TSBADR	JSBADR	PROG	FILE
FILEMGR	0502	90A4	C08A	209F	0005	F320		91F4	71A4	S\$SHIP	
SYNCOB	0301	6CE4	476A	C5CF	DE04	0C00	9	9152	71A4	PROG	
FILEMGR	0502	90A4	C08A	209F	0005	F320		91F4	717A	S\$SHIP	
SYNCOB	0301	6CE4	476A	C5CF	DE04	0C00	8	9152	717A	PROG	
FILEMGR	0502	90A4	C08A	209F	0005	F320		91F4	7126	S\$SHIP	
SYNCOB	0301	6CE4	476A	C5CF	DE04	0C00	6	9152	7126	PROG	
FILEMGR	0502	90A4	C08A	209F	0005	F320		91F4	70FC	S\$SHIP	
SYNCOB	0301	6CE4	476A	C5CF	DE04	0C00	5	9152	70FC	PROG	
RPRCP	4C13	E27E	E25E	209F	0004	C520		9410	6F88	S\$UTIL	
FILEMGR	0502	90A4	C04C	249F	0024	F320		91F4	6F88	S\$SHIP	
SCI990	0101	9D78	5680	21CF	AC06	3000	39	9152	6F88	S\$UTIL	
ANALZ	1A12	7F06	17E2	C4CF	AA04	4400	39	9368	6F88	S\$UTIL	
XOI	620D	597A	1998	25CF	A906	3000	39	92B0	6F88	S\$UTIL	
FILEMGR	0502	90A4	C04C	249F	0024	D320		920E	6D96	S\$SHIP	
SCI990	0101	7EC6	7C78	21CF	D005	1000		9152	6D96	S\$UTIL	
LGACCT	47F4	E7EC	C188	C49F	0005	F300		A006	4C9E	S\$UTIL	
LGFORM	21E2	FC5C	FC3C	209F	0004	C700		9B92	4C9E	S\$UTIL	
PMRWTK	2DD8	C1D0	C008	B49F	7904	C500		9A3C	4C9E	S\$UTIL	
JOBMGR	48D7	E0FC	C1F6	C49F	0004	C720		99CA	4C9E	S\$UTIL	
DISKMGR	0619	C006	C02C	B49F	0024	F300		98C8	4C9E	S\$SHIP	
PMWRIT	0817	C006	C082	B09F	0024	F320		9856	4C9E	S\$SHIP	
IPC	0614	C232	C022	B09F	0024	D100		97E4	4C9E	S\$UTIL	
PMDVYL	0713	C006	C03A	B49F	0024	D120		9772	4C9E	S\$SHIP	
SAVRES	680D	C008	C09E	B09F	0024	D300		9570	4C9E	S\$UTIL	
FILEMGR	0509	90A4	C04C	209F	0024	F320		94FE	4C9E	S\$SHIP	
PMTERM	0408	C006	C264	D89F	0005	F120		9476	4C9E	S\$UTIL	
NAMMGR	4306	C00A	C0A0	B09F	0024	F120		937C	4C9E	S\$UTIL	
IDU	0305	C006	C030	B49F	0024	F320		92F4	4C9E	S\$SHIP	
PMTLDR	0404	C006	C18C	309F	0024	F320		9282	4C9E	S\$SHIP	
DIDU	7502	C0B2	C012	B49F	0024	F120		919E	4C9E	S\$UTIL	
OPERATOR	6111	9022	C14E	959F	7909	9008		9656	4C9E	S\$UTIL	
MAILBOX	0710	1ADA	0588	C5CF	7909	1008		9700	4C9E	S\$UTIL	
PMTBID	0203	C006	C036	B09F	0024	F320		9210	4C9E	S\$SHIP	

Figure 8-3. Task States and JSB List (Sheet 1 of 2)

```

**** JSB LIST ****

71A4-717A 000B 6002 1404 00FF 0000 0000 0000 .. .. .. .. ..
71B4-A0C6 4CDA 5439 2020 2020 2020 2020 2020 .. L. T9
71C4-2020 2020 0000 71A4 .. ..

717A-7126 000A 6002 1404 00FF 70FC 0000 0000 .& .. .. .. ..
718A-A00B 4CDA 543B 2020 2020 2020 2020 2020 .. L. TB
719A-2020 2020 0000 717A .. ..

7126-70FC 000B 6002 1404 00FF 71A4 70FC 0000 .. .. .. ..
7136-9EAE 4CDA 5436 2020 2020 2020 2020 2020 .. L. T6
7146-2020 2020 0000 7126 .. .&

70FC-6F88 0007 6002 1404 00FF 7126 717A 0000 .. .. .. .. .& ..
710C-9E12 4CDA 5435 2020 2020 2020 2020 2020 .. L. T5
711C-2020 2020 0000 70FC .. ..

6F88-6D96 0002 4005 0C02 A AFF 0000 0000 0000 .. .. @. .. ..
6F98-9688 4CDA 4F20 2020 2020 2020 5359 5354 .. L. 0 SY ST
6FAB-454D 2020 0000 6F88 EM .. ..

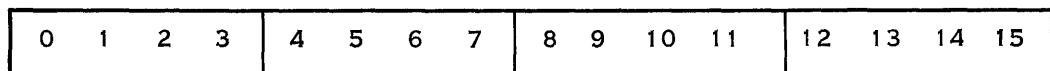
6D96-4C9E 0001 4002 1402 D0FF 70FC 0000 0000 L. .. @. .. ..
6DA6-93F4 4CDA 5359 5324 494E 4954 2020 2020 .. L. SY S$ IN IT
6DB6-2020 2020 0000 6D96 .. ..

4C9E-0000 0000 801E 0002 00FF 0000 0000 0000 .. .. .. ..
4CAE-9066 4CDA 5324 5348 4950 2020 2020 2020 .. L. S$ SH IP
4CBE-2020 2020 0000 ..
    
```

Figure 8-3. Task States and JSB List (Sheet 2 of 2)

The following paragraphs briefly describe each entry of the ANALZ listing shown in Figure 8-3.

8.2.2.1 Task States. For each task in the system, the task state list contains the task ID, the task context at the last time the task was scheduled or performed an SVC, the current state of the task, the task flags, the TSB address, the JSB address, and the program file name. You may need a list of program files to identify these tasks. The task flag indicates task characteristics. Figure 8-4 defines the flags in the flag word.



2279433

- 0 — System task
- 1 — Privileged task
- 2 — Current segment set in memory
- 3 — Take end action on error
- 4 — I/O has been aborted for task
- 5 — Task being aborted
- 6 — Bypass security
- 7 — Queue server task
- 8 — Activate task outstanding
- 9 — Initial task bid
- 10 — Software privileged task

Figure 8-4. Bit Assignment for the Flag Word

Bit 2 of the task flag contains information for the active segment of the task at the time of the crash. When this bit is reset, the segment of the task has been rolled out to the disk. By examining this bit for each task, the systems programmer can determine which tasks were in memory and may be associated with the crash.

8.2.2.2 JSB List. The JSB list contains all global data about the job including the job ID, job name, and priority. This information allows the systems programmer to associate a job with the task that was executing at the time of the crash.

8.2.3 Analyzing System Crash Dumps

The most common types of crashes are >6x (> 60 through >6F) crashes and >1x (> 10 through >1F) crashes. The following paragraphs give suggestions on conditions to look for when analyzing these crashes.

The >6x crash is caused by an invalid internal interrupt within the system or within a system task. Therefore, the status register at the time of failure is >xxx1. The following steps indicate what to look for when a >6x crash occurs.

Register 1 of the interrupt level 2 workspace contains the task error code that caused the crash. This code is displayed as a part of the crash code. The contents of registers 13, 14, and 15 show the location of the crash and the status at the time of the crash. If bit 8 of workspace register 15 is set to 1, the crash occurred in map file 1. If it is set to 0, the crash occurred in map file 0 (in system code). Register R14 contains the program counter value at the time of the crash. The 16 memory locations around the program counter value are also printed. Therefore, the systems programmer can decode the instructions previous to the program counter value (which was incremented by 2). The code around this location indicates the cause of the crash. The systems programmer can use the task workspace to check indexed addresses. For example, if a crash code is > 65 (a memory mapping violation), the systems programmer should decode the instructions using the task workspace register information.

When a task takes end action, the TSB for the task contains the address of a four-word area that is printed following the TSB. These words contain the error code and the context of the task at the time end action was taken. In the case of end action taken by a system task, the systems programmer should use this information to determine what forced a task to take end action. The systems programmer can then decode the location previous to this program counter value.

The > 1x crash is an illegal interrupt at interrupt level x. This indicates that a device interrupted at an interrupt level that is not defined or a device interrupted from an expansion chassis and its position was not defined in the chassis. Check the hardware configuration and the system configuration listing.

8.2.4 Forcing a System Crash

A problem can occur in the system without resulting in a system crash, but it might prevent useful work on the system. For example, a system task could be in an endless loop. In such a case, it is desirable to force a system crash to obtain a crash dump for analysis.

To force a crash, perform the following steps:

1. Press HALT twice to ensure that the processor is using map file 0.
2. Press PC DISPLAY, MA ENTER, CLR, and MDE, in that order, to set current memory address to 0.
3. Press RUN to resume execution. The machine instruction code executed at this point is > 0000, entered from the programmer panel. This is an illegal instruction and causes the system to crash with crash code > 62.
4. Now, copy the crash dump to the crash file by pressing HALT and RUN.

When you force the system to crash, the immediate cause of the crash is the illegal instruction you forced the computer to attempt to execute. However, you can examine other information (such as the executing task, hardware trap vectors, XOP trap vectors) to determine what caused the problem. The systems programmer should look for the location where the crash occurred. This location can then be used to trace the task that was executing when the problem occurred.

Adding Error Messages

9.1 DNOS MESSAGE AND ERROR PROCESSING

DNOS uses a single format for error messages from the operating system and all utilities. The method DNOS uses for message handling has the following features:

- It provides informative, consistent messages.
- It allows you to easily find expanded explanations of the errors.
- It permits you to add your own messages.

DNOS offers the following three levels of error reporting:

- Internal error codes
- Short, descriptive, English messages with message categories and identifiers used to find further explanation
- Expanded online explanations

This section describes the addition of messages and expanded explanations.

Internal error codes appear if the supplied directories of message files are deleted from the system disk. If the directory `.$MSG` is on the system disk, short messages are supplied. If the directory `.$EXPMSG` is also on the system disk and key indexed file (KIF) support is available, expanded explanations for errors are also available.

9.1.1 Internal Error Codes

When only the first level of error reporting is available, DNOS generates error messages formatted as follows:

```
CCCCCCCC—INTERNAL CODE >MMM      VVVVVVVV
```

Message category CCCCCCC is a one- to eight-character string. The categories used by DNOS are as follows:

Category	Meaning
ASSEMBLR	Macroassembler messages
DEBUGGER	Task Debugger messages
DNOSHLL	DNOS high-level language messages
EDITOR	Text Editor messages
LINKER	Link Editor messages
MAIL	Used by mailbox facility
SCI	System Command Interpreter (SCI) messages
STATUS	Task status messages
SVC	Supervisor call (SVC) messages
UTILITY	Utility messages

Displaying the message category CCCCCCC is optional. Refer to Section 3 for a description of the S\$CMMSG interface routine.

Error code MMMM is a four-digit, hexadecimal, ASCII-character internal error code.

Variable text string VVVVVVVV is an optional, variable-length text string with logically separate pieces separated by semicolons.

9.1.2 Short English Messages

When the second level of error reporting is available, error messages have the following format:

SSS CCCCCCC—NNNN message message message message message message message message message message message (as many as five lines)

Displaying the error source code SSS and message category CCCCCCC is optional. Error source code SSS contains one, two, or three characters. The following is a list of error source code characters:

Code	Meaning
H	Hardware
I	Informative
S	System
U	User
W	Warning

Message category CCCCCCC is one of the abbreviations defined for internal error codes.

Additional abbreviations are used for support of various programming language and productivity tools. The abbreviations used by each language are described in the appropriate manual for the language.

Message number NNNN can be used to locate further explanation of the message in the section for category CCCCCCC within the *DNOS Messages and Codes Reference Manual*.

9.1.3 Expanded Explanations Online

For those systems that support expanded message files, the following information is reported to the user on request:

Explanation: Text that explains the probable cause of the error and what the system has done

User Action: Text that explains what the user should do to recover from the condition

When an error message is displayed, enter a question mark (?) and press the Return key to display the expanded error message.

9.1.4 Show Expanded Message (SEM) Utility

The SEM utility displays an expanded explanation of a specified error message. You can display the expanded explanation of an error message immediately after the message appears by entering a question mark (?), and pressing the Return key. At any other time, you must enter an SEM command to display the expanded explanation. The information is written to the terminal local file (TLF) and displayed. Enter the command as follows:

```
SHOW EXPANDED MESSAGE
  MESSAGE CATEGORY: alphanumeric
                MESSAGE ID: [{alphanumeric/integer}]
  INTERNAL ERROR CODE: UNKNOWN/integer
```

The message category is the abbreviation for the category of the desired expanded message file. A message can be retrieved according to the message identifier sent to the screen or by the internal error code. A message identifier is the message number sent to the terminal that immediately follows the hyphen in the error message. An internal error code is the hexadecimal value used internally by DNOS. A common use of the INTERNAL ERROR CODE prompt is to display the message for an error returned to a supervisor call (SVC) block.

For example, the user might specify the following to display the explanation of the SVC error shown for the Assign LUNO Error, numbered 0315:

```
SHOW EXPANDED MESSAGE
  MESSAGE CATEGORY: SVC
                MESSAGE ID: 0315
  INTERNAL ERROR CODE: UNKNOWN
```

9.1.5 Displaying Messages

The message is a combination of file-resident text and variable text. The file-resident text can contain as many as 240 characters. Any character other than the question mark can appear in the text. The question mark is used as a position marker. It is replaced by variable text when the message is processed. Each question mark is followed by a decimal digit between one and nine to show which variable text element replaces the question mark. A question mark (?) followed by the character C implies that the current line of message text is filled with blanks; that is, C is an effective carriage return/line feed sequence.

Variable text is that part of the displayed message that can vary in each display of the message. It is determined at run time. It can be a pathname, logical unit number (LUNO), opcode, or other execution-dependent information.

When the message is being formatted for display, the disk-resident portion of the message is placed in the message output buffer. Each question mark is replaced by variable text supplied by the task that detected the error. The maximum length of variable text is 235 characters. This includes all variable text elements and delimiters that may be in the buffer.

One question mark and digit is used for each element of variable text. When an element of variable text is null, the question mark remains in the message but the associated digit is removed. An element can contain as many as 235 characters.

The variable text delimiter is the semicolon. More than one variable text element can be used in the variable text message. Two consecutive semicolons can be used to represent a null variable text element.

If the files containing the file-resident portion of the message text are not on the system disk, the internal error code (previously described) is displayed.

9.1.6 Message Examples

The following examples show typical internal error codes and error messages for two error conditions. The internal codes, message file texts, and message numbers shown in these examples are not necessarily those currently in the system. They are shown to illustrate the formatting of DNOS error messages.

9.1.6.1 Assign LUNO Error. When the file specified in an Assign LUNO operation does not exist, the internal error code displayed on a system without a message text file might be as follows:

```
SVC — INTERNAL ERROR CODE >0027 .PRINT.OUT
```

The message category is SVC, the internal error code is >0027, and the variable text is the path-name of the nonexistent file .PRINT.OUT. When a message text file is present, an error message is displayed. The following is a sample error message for the same error:

```
U SVC - FILE .PRINT.OUT DOES NOT EXIST
```

The U identifies this as a user error, the message category is the same as that used in the internal error code, and the message is a composite of the message text and the variable text. The message text is as follows:

```
FILE ?1 DOES NOT EXIST
```

The variable text is as follows:

```
10.PRINT.OUT
```

The character count is a binary value. The characters replaced the ?1 of the message text.

A message ID in the error can be used to find additional information in the *DNOS Messages and Codes Reference Manual*. It is also the number that identifies a supplementary message in the expanded message file. The following message example contains a message ID:

```
U SVC — 0315 FILE .PRINT.OUT DOES NOT EXIST
```

9.1.6.2 COBOL Compiler Termination. The following is a sample display of the internal code displayed on a system without a message text file when the COBOL compiler has satisfactorily completed:

```
COBOL - INTERNAL CODE >9010 .SOURCE;0
```

The message category is COBOL. The internal error code is >9010. The variable text consists of the pathname of the source file, .SOURCE, and the number of errors the compiler detected, 0.

When a message text file is present, an error message is displayed. The error message for COBOL completion could be:

```
I COBOL — .SOURCE COMPILED WITH 0 ERRORS
```

The I identifies this as an Informative message. The message category is the same as that displayed in the internal message code, and the message is a composite of the message text and the variable text. The message text is as follows:

```
?1 COMPILED WITH ?2 ERRORS
```

The variable text is as follows:

```
9 .SOURCE;0
```

The character count is a binary value. The characters of the first element (up to the semicolon) replaced the ?1 of the message text. The 0 replaced the ?2.

9.2 MESSAGE FILES

Two types of message files can be maintained on the system disk:

- Message files in the .S\$MSG directory provide the fixed text portion of the error, information, and completion messages used by SCI, SVCs, the languages, and utilities.
- Expanded message files in the .S\$EXPMSG directory contain the expanded explanations that are displayed when requested.

Directory .S\$MSG contains a message file for each of the language processors, the major utilities, and the SVC processors. The Build Message File (BMF) utility allows users to create message files for use in .S\$MSG.

Directory .S\$EXPMSG contains an expanded message file corresponding to each message file. These files contain expanded explanations of the errors documented in the message files. The BEMF utility allows users to create expanded message files for use in the directory .S\$EXPMSG.

The messages files required by various application processors and utilities should use file names unique to those processors. The file name can be any name except those reserved for the system communications product and language processors. The following names are reserved:

ASSEMBLR	EDITOR	NIO	SMRG
BASIC	FORTRAN	PASCAL	STATUS
COBOL	FORT78CP	PTP	SVC
COMM	FORT78RT	QUERY	TAP
DATADICT	ICS3270	RPG	TIFORM
DBMS	LAN	S\$ROUTIN	TIP
DEBUGGER	LINKER	SCI	TIPE
DNCS	MAIL	SLA	UTILITY
DNOSHLL			

To enable flexible naming for files, the source code contains a file indicator instead of the file name. The system uses the file indicator (hexadecimal value between >00 and >7F). This file indicator is associated with a file name by a synonym \$\$FNxx in the SCI command procedure that calls the application processor.

Application programs written by users can use files with file indicators greater than >7F. Any conflict between file indicators in user programs is resolved by avoiding any common files or command procedures.

9.2.1 Format of the Message Text Files

The message file is written by a utility called by the Build Message File (BMF) command. The input to this utility is a blank-suppressed sequential file. A file written by the Text Editor with default parameters is the intended input. A record length other than the default 80-character record length can be used. The following paragraphs describe the file contents.

The first record of a message text file is a heading line. It must contain the following information:

- The lowest-value internal error code in the file in columns 1 through 4 (4 digits)
- The highest-value internal error code in the file in columns 6 through 9 (4 digits)
- The local language character for U (user error) in column 11
- The local language character for S (system error) in column 12
- The local language character for H (hardware error) in column 13
- The local language character for W (warning) in column 14
- The local language character for I (informative) in column 15 (the second record should be a blank line)

Individual messages begin in record 3. Each message in the file has a header containing the following fields:

- The error source indicator or combinations (such as USH)
- One or more blanks
- A left parenthesis
- One or more four-digit hexadecimal internal error codes (separated by commas)
- A right parenthesis

This header can extend to several 80-character lines when the applicable internal error codes cannot all be written on one line. Terminate each line with the comma that separates the last internal error code on the line from the first code on the next line.

The text of the message begins in column 1 of the next line and can be up to three lines long. This provides a maximum of 240 characters for the fixed-text portion of a message. The message IO, when it is to be displayed, precedes the message text on the first line and is included in the maximum number of characters. The message text is followed by a blank line. A text message can contain any printable characters, with the following restrictions:

- A question mark (and digit) anywhere in the message is replaced with variable text supplied by the task that reported the error.
- The question mark and digit pair can be set off by blanks, have a blank on one side, or have nonblank characters on both sides.
- A message can include as many as nine question marks, but none are required.

A message can be associated with many internal error codes or with only one. The internal error code, perhaps by means of an EQU directive, appears in the source code of the task that issues an error message rather than the message itself. This allows you to add, modify, or delete messages without changing the source code.

The internal error codes used within one message text file are independent of those used in another file. The user can choose any range of values. The values should be continuous to minimize file space required since the .S\$MSG file built from the message text file includes a continuous table of internal message numbers and their corresponding record numbers in the .S\$MSG file.

The standard message text files found in the .MESSAGES.TEXT directory have messages in uppercase English.

The following is an example of a message text file:

```
9050 9070 USHWI

SU (9050)
0059 ERROR ENCOUNTERED IN ASCII CONVERSION

U (9052,9070)
0061 ATTEMPT TO MODIFY BYTE AT AN ODD ADDRESS

I (9053)
0062 MODIFICATION PREVIOUSLY APPLIED
```

The following example shows a message text using this file and internal error code 9051. The message category UTILITY is determined by selecting the file. The error source code S is supplied by the task that issues the message. The complete message displayed is as follows:

```
S UTILITY—0060 INTERNAL SUBROUTINE ERROR ENCOUNTERED
```

9.2.2 Format of the Expanded Error Message Text Files

The expanded message file is written by a utility called by the Build Expanded Message File (BEMF) command described in paragraph 9.3.2. The input to this utility is a blank-suppressed sequential file of 80-character records (normal Text Editor output), as described in the following paragraphs.

Each message file can have a companion expanded message file in the expanded message file directory. This file should have the same name as its counterpart in the message file directory. These files contain the explanation and user action portions of the messages that appear in the *DNOS Messages and Codes Reference Manual*. The files for system messages contain text in uppercase and lowercase.

Record 0 of an expanded message text file contains the characters for the headings for the messages (Explanation and User Action). These are in the same language as the messages in the file. Each heading must be enclosed in quotation marks to allow blanks in the phrases when appropriate.

Subsequent records contain an explanation message and a user action message. For each pair of messages, the records contain the following:

1. The characters %% are followed by a message identifier of up to 14 characters. The identifier must be the same as the external message number specified for this message in the .TEXT file. The pairs %%1 and %%2 are reserved for the system.
2. Starting on a new line, a paragraph explaining the message. The paragraph can contain as many as 20 lines.
3. A blank line.

4. A paragraph describing user action when the message is seen. The paragraph can contain as many as 20 lines.
5. A blank line.

Expanded explanations should be included for all of the entries in the message text file that contain message IDs. The standard expanded message text files are found in the .MESSAGES.EXPTXT directory. The following example shows an expanded message text file that corresponds to the example .TEXT file in the previous section.

%%0059

The attempt to convert to ASCII resulted in an error.

Examine the program and determine why the input for the conversion was in error.

%%0061

The address specified for the change was an odd address. This routine does not process addresses with an odd value.

Determine the source of the address, correct the error, and try the operation again.

%%0062

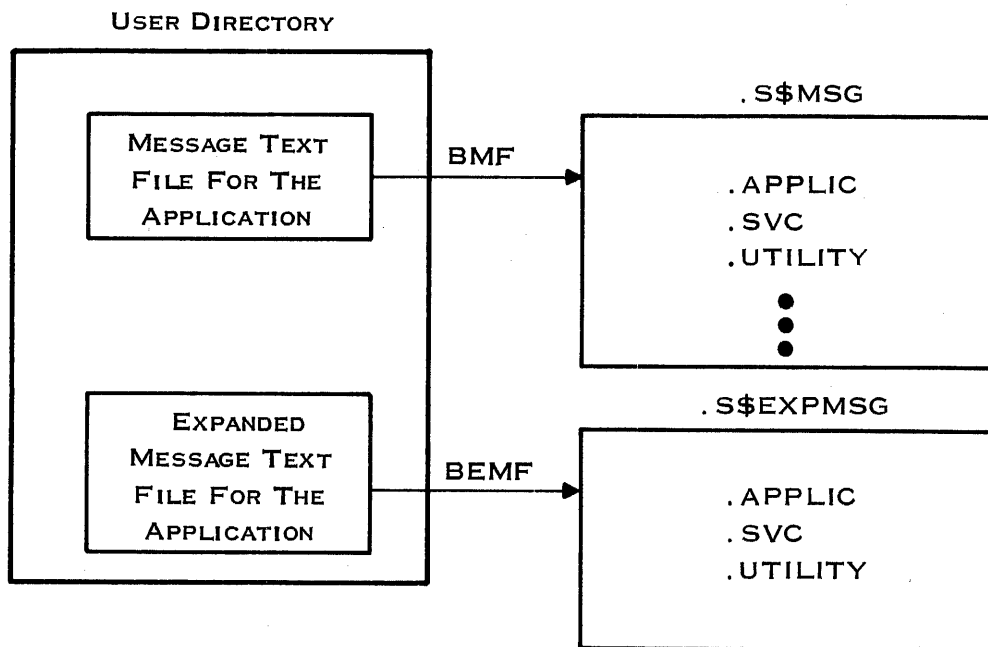
The requested modification has already been made.

This is an informative message only.

9.3 MESSAGE FILE UTILITIES

Three utilities are provided to build and use the message files. One of these utilities builds the message files from the message text files. Another creates expanded message files from the expanded message text files. A third utility retrieves the expanded explanation for the message ID given by the user.

Each of these utilities is called by an SCI command. Figure 9-1 shows the functions of the utilities that process the message text files and the expanded message text files.



2279434

Figure 9-1. Functions of Message File Utilities

9.3.1 Message File Utility

To create a message file in the .S\$MSG directory, first write a message text file in any user directory. Then, use a Build Message File (BMF) command to write the file in the .S\$MSG directory. The command is as follows:

```

BUILD MESSAGE FILE
      INPUT FILENAME: filename@
      OUTPUT FILENAME: filename@
      OUTPUT FILE TYPE(REL/SEQ): REL/SEQ      (REL)
      ERROR ACCESS NAME: filename@      (DUMY)
      MAXIMUM MESSAGE TEXT LENGTH: integer      (80)
  
```

The input file name is the file pathname of the message text file that contains error messages for user programs. The output file name is the file name that must be placed in the .S\$MSG directory for use by DNOS. The error access name is the pathname of a listing file used to document any errors in the message text file. If errors occur, correct them and reenter the command. The output file can be either a sequential or a relative record file. A relative record file (the default file) is accessed in less time. However, you can specify a sequential file when file size is important. The sequential file is more compact. The maximum message text length is the logical record length of the message text file specified as the input file. It is normally 80 characters. No translation of lowercase to uppercase is available. It is expected that the message text file contains uppercase letters. The .S\$MSG file is also in uppercase letters.

9.3.2 Expanded Message File Utility

SCI can display any expanded explanation of the errors documented in the user message files. The expanded message text file can be developed in any user directory. You can then prepare this file for directory `.$$EXPMSG` by using the Build Expanded Message File (BEMF) command. Files in directory `.$$EXPMSG` are KIFs.

```

BUILD EXPANDED MESSAGE FILE
      INPUT FILENAME: filename@
      OUTPUT FILENAME: filename@
      ERROR ACCESS NAME: filename@ (DUMMY)
CONVERT LOWER TO UPPER CASE?: YES/NO (NO)
      MAXIMUM MESSAGE ID LENGTH: integer (4)

```

The input file name is the file pathname of the expanded text file. The output file name is the name of the file that must be placed into the `.$$EXPMSG` directory for use by DNOS. The error access file is the file pathname of a file that is used to document any errors in the expanded message file. When errors occur, delete the output file. Correct the expanded message file and reenter the BEMF command.

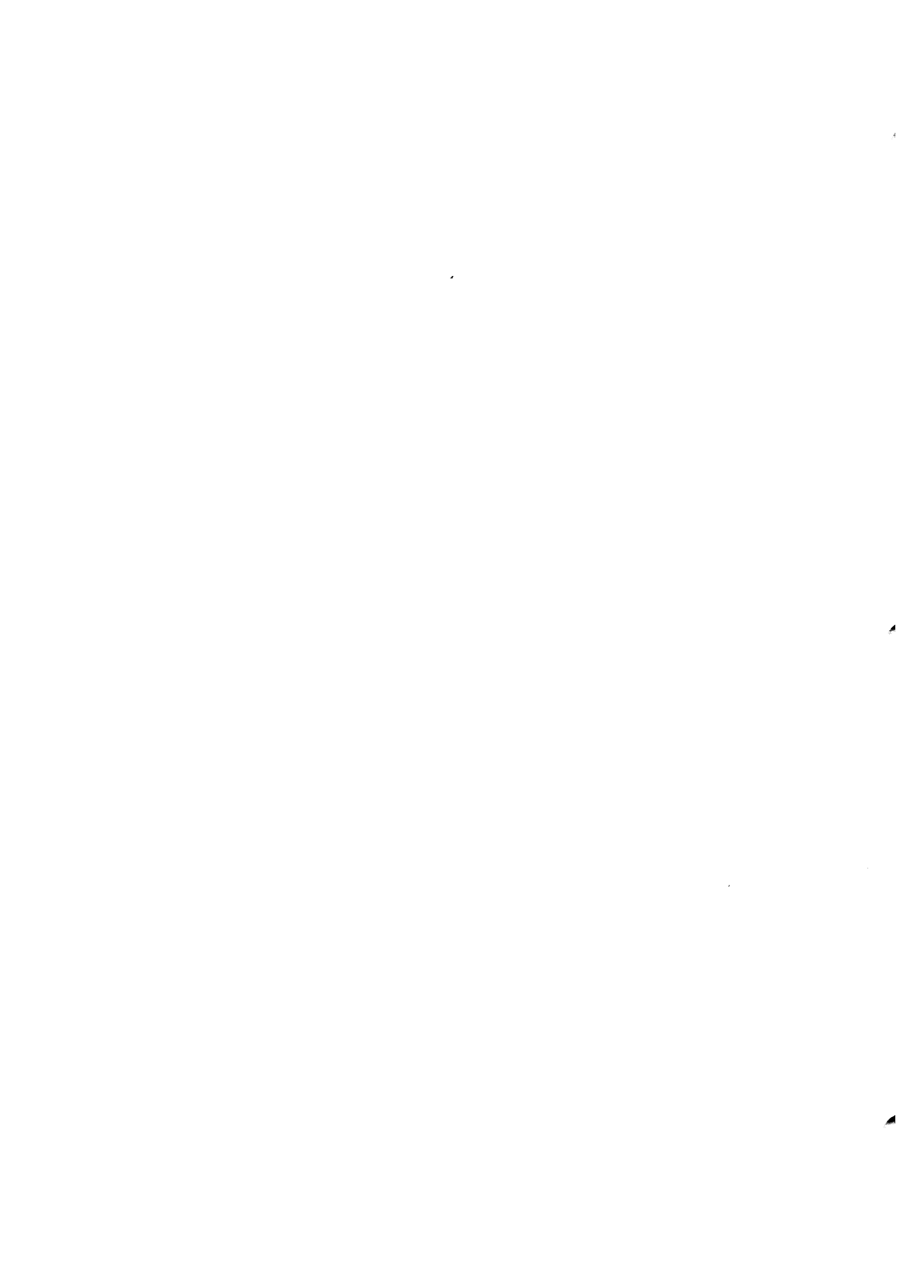
Respond YES to the CONVERT TO UPPERCASE prompt when the system includes terminals that do not support lowercase characters and the input file contains lowercase letters. Otherwise, you can respond NO to the prompt.

The maximum message ID length is the length of the message number that is used to select the expanded explanation. The default value is 4. You can specify any value from 1 through 14.

9.4 ERROR MESSAGE INTERFACE

An application program uses routines `S$TERM` and `S$CMMSG` (described in Section 3) to interface with the message files. Section 3 also describes the system synonyms related to these routines (`$$CC` and others).

Command procedures that use the application message file need to set a synonym `$$FNxy` to the message file name in use. The value chosen for `xy` must be greater than `> 7F` and must not conflict with any other application message file in use. The value assigned to the synonym must be the last component of the file name in the `.$$MSG` directory. For example, an accounting package might have a file `.$$MSG.ACCOUNT` for which the synonym `$$FN80` is set to `ACCOUNT` in each command procedure used by the accounting application.



International Considerations

10.1 INTRODUCTION

DNOS is an international operating system designed to meet the commercial requirements of the United States as well as most Western European countries and Japan.

The international capabilities include the following:

- Data input and output via international peripheral devices designed to meet the language requirements of the following countries:

- Denmark/Norway
- France/Belgium
- French word processing
- Germany/Austria
- Japan
- United States
- United Kingdom
- Sweden/Finland
- Spain
- Switzerland

- The ability to store and manipulate international data on any of the DNOS file types
- Collating sequences dependent on country code for key indexed files (KIFs)
- Translatable error messages
- Translatable of SCI procedures
- Use of special language characters in pathnames and synonyms

10.2 COUNTRY CODE

DNOS interprets international data in accordance with the country code selected by the user during system generation. This code reflects the nationality of the data terminals attached to the system and defines the way data is processed for the complete user system. It also determines the way the device service routines (DSRs) interpret input and output data and the way in which (KIF) management orders user keys.

The country codes that can be assigned during system generation are as follows:

Response to Country Code Prompt	Country
AU	Austria
B	Belgium
D	Denmark
FI	Finland
FRA	France
FWP	French Word Processing
G	Germany
J	Japan
N	Norway
SP	Spain
SWE	Sweden
SWI	Switzerland
UK	United Kingdom
US	United States

NOTE

For countries not listed, users should enter the default value (US) for the country code prompt during system generation.

When using SCI, you can use the Show Country Code (SCC) command to show which country code is currently set. Table 10-1 shows the ASCII codes for special language characters. User programs can determine the country code assigned to their system by executing a Retrieve System Information (> 3F) supervisor call (SVC).

Table 10-1. ASCII Codes for Special Language Characters

COUNTRY	ASCII CODE										
	23	40	5B	5C	5D	5E	60	7B	7C	7D	7E
US ASCII STANDARD	#	@	[\]	^	`	{	:	}	~
UNITED KINGDOM	£	@	[\]	^	`	{	:	}	~
GERMANY/AUSTRIA	#	@	Ä	Ö	Ü	^	`	ä	ö	ü	ß
SWEDEN/FINLAND *	#	É	Ä	Ö	Å	Ü	é	ä	ö	å	ü
NORWAY/DENMARK	#	@	Æ	Ø	Å	^	`	æ	ø	å	~
SPANISH-SPEAKING	#	@	í	Ñ	¿	^	`	°	ñ	ç	~
SWITZERLAND	£	à	é	ç	è	^	`	ä	ö	ü	..
FRANCE **											
FRENCH WP	£	à	°	ç	§	^	`	é	ù	è	..
ITALY **											
HOLLAND **											

*↔ USES ASCII CODE >24, REPLACING THE \$ CHARACTER IN US ASCII.

** USES THE US ASCII STANDARD.

2284937(1/2)

10.3 INFORMATION INTERCHANGE CODES

DNOS and its supported peripheral devices use the internationally accepted information interchange codes for the various countries supported. In most cases, this is a seven-bit ASCII-type code with a few special characters required for the local language.

Japan uses an eight-bit code to represent both the Latin alphabet (A through Z) and the Japanese Katakana character set in one combined information interchange code (JISCII). Devices using this code must be put in 8-bit mode by using the Modify Device State (MDS) command. For details about the MDS command, refer to the *DNOS System Command Interpreter (SCI) Reference Manual*. Table 10-2 shows the JISCII codes for Japanese characters.

The Japanese Model 911 VDT, which supports this extended character set, does not have the high/low intensity features available on the other versions of the Model 911.

Table 10-2. JISCII Codes for Japanese Characters

JISCII CODE	5C	A1	A2	A3	A4	A5	A6	A7	A8	A9	AA	AB	AC
JAPANESE CHARACTER	キ	ク	ケ	コ	カ	キ	ク	ケ	コ	カ	キ	ク	ケ
JISCII CODE	AD	AE	AF	B0	B1	B2	B3	B4	B5	B6	B7	B8	B9
JAPANESE CHARACTER	ユ	ヨ	ユ	ー	ア	イ	ウ	エ	オ	カ	キ	ク	ケ
JISCII CODE	BA	BB	BC	BD	BE	BF	C0	C1	C2	C3	C4	C5	C6
JAPANESE CHARACTER	コ	ク	シ	ス	セ	ソ	タ	チ	ツ	テ	ト	ナ	ニ
JISCII CODE	C7	C8	C9	CA	CB	CC	CD	CE	CF	D0	D1	D2	D3
JAPANESE CHARACTER	ス	ネ	ノ	ハ	ヒ	フ	フ	ホ	マ	メ	ム	モ	モ
JISCII CODE	D4	D5	D6	D7	D8	D9	DA	DB	DC	DD	DE	DF	
JAPANESE CHARACTER	ヤ	ユ	ヨ	ラ	リ	ル	レ	ロ	ワ	ン	ニ	ノ	

2284937(2/2)

Local character extensions to ASCII required for international ASCII often replace special characters. The brackets ([]) used for the SCI prompt may not be available; local characters may have replaced them. For example, in German, the letters A U replace the brackets in the default SCI prompt. You can supply an appropriate substitute by entering the .OPTION SCI primitive. The format of the primitive is as follows:

.OPTION PROMPT = "Any message"

10.4 KIF COLLATING SEQUENCES

The KIF manager sorts user keys alphabetically (rather than according to the hexadecimal value of the letter) as it inserts records in a KIF. In German, Austrian, Swedish, Danish, Norwegian, and Finnish, the alphabetic order differs from English and a special collating sequence applies.

The country code specified for the system determines the collating sequence for KIFs. For example, the key letters V, Ü, U, and D are sorted by KIF as D, U, Ü, and V in a German system.

Key Order on Disk (German)	Hexadecimal Value
D	44
U	55
Ü	5D
V	56

Once a KIF is written on a German system, the collating sequence in which the data is arranged is valid only for a German system. For example, if the file is physically transferred to a Swedish system and a Show File (SF) command is executed, the data appears reasonably legible, although the U appears as an A on the Swedish VDT. However, the alphabetic sorting order of the keys is incorrect for the Swedish system. The Swedish A should not appear between the U and the V in the collated access list. If you attempt to add data to the file, the key collating sequence is disrupted and the file becomes unusable.

If you use binary data as keys, you may have several values that map to the same positions in sorted order. At most, the last four byte values of the ASCII codes will sort to the same value; that is, > FC, > FD, > FE, and > FF will all sort at the end as > FF.

To successfully transfer a KIF between different international systems, convert the file to a sequential file by using the Copy KIF to Sequential File (CKS) command on the system on which the file was written. Then, create the KIF on the destination system. Use the Copy Sequential File to KIF (CSK) command to copy the records using the collating sequence of the destination system. Since the new KIF is compatible with the destination system, you can insert records correctly using the CSK command.

Table 10-3 shows the collating sequences for all supported languages.

Table 10-3. Collating Sequences for All Supported Languages

Country	Collating Sequence
France/Belgium	ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz
France Word Processing	ABCDEFGHIJKLMNOPQRSTUVWXYZ aàbcçdeéèfghijklmnopqrstuùvwxyz
Germany/Austria	AÄBCDEFGHIJKLMNOPÖPQRSTUVWXYZ aäbcdefghijklmnoöpqr sβtuüvwxyz
Japan (Katakana)	ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz ファイウエオヤユヨッーア イウエオカキクケコサシス セン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	ABCDEÉFGHIJKLMNOPQRSTUVWXYZÜZÅÄÖ abcdeéfg hijklmnopqr stuvwxyzüzåäö
Spanish-speaking countries	ABCDEFGHIJKLMNÑOPQRSTUVWXYZ abcçdefghijklmnñopqr stuvwxyz
Switzerland	ABCDEFGHIJKLMNOPQRSTUVWXYZ aäåbcçdeéèfghijklmnoopqrstuüvwxyz
United Kingdom	ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz

228 30 37

10.5 INTERNATIONALIZING MESSAGES

Section 9 describes the DNOS message facility in detail. The messages reside in a set of files, which facilitates translating them into the local language.

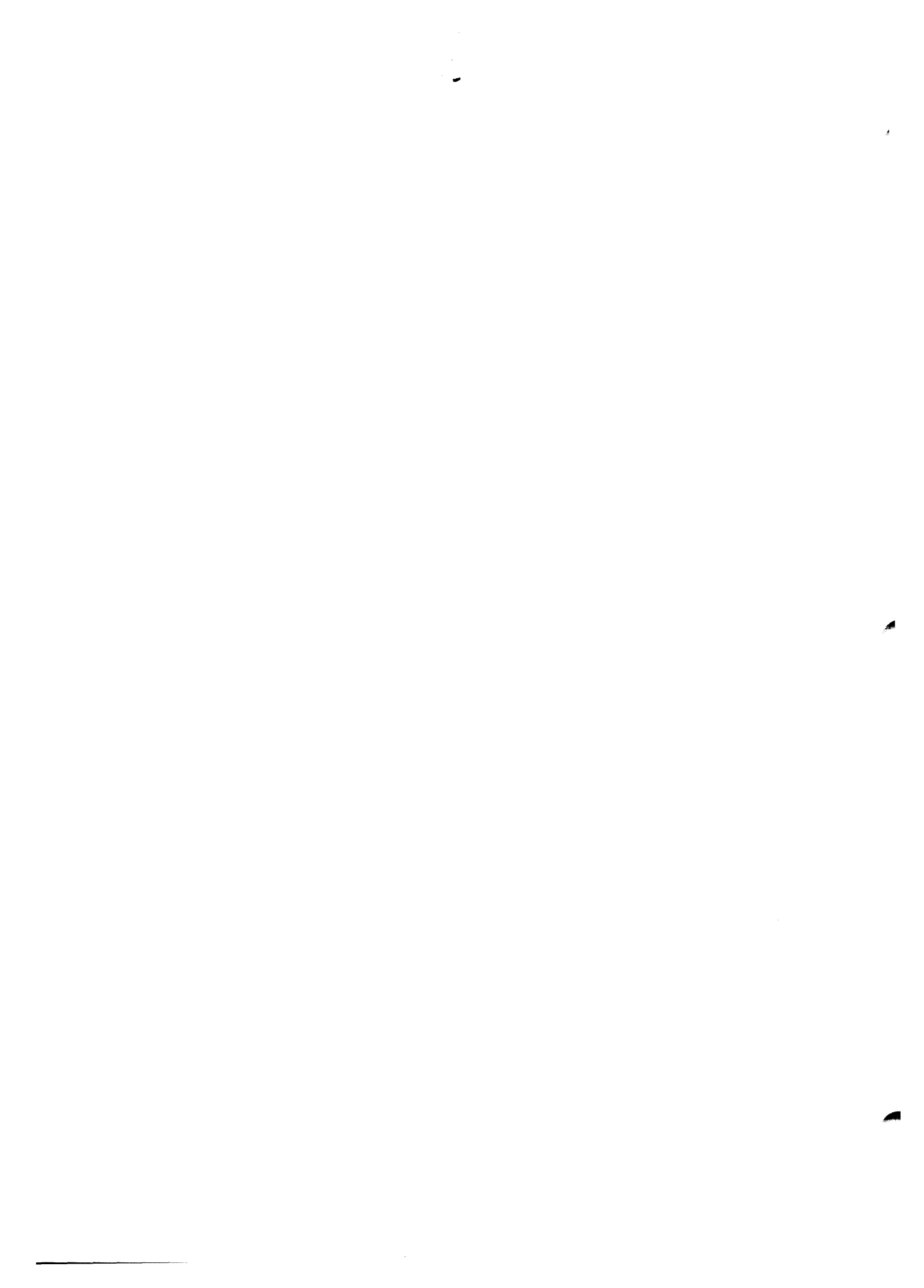
The `.MESSAGES.TEXT` directory contains the files of factory-built text messages in a form that can be text edited. Separate files are available for each of the language processors, System Command Interpreter (SCI), the major utilities, SVC error messages, and user-developed messages.

The header record (first record) of each file contains the local language letters that correspond to the error source code letters U, S, H, W, and I. (See the paragraph entitled Short English Messages in Section 9 for the meanings of the error source code letters.) Edit that record to place the proper letters in columns 11 through 15 as described in the section on message files.

For each error message, the `.MESSAGES.TEXT` directory contains edit files with source code, the internal error codes that select the message, and the message text. Edit the error source code field, substituting the letters defined in the header record for those in the file. Then edit the message text, replacing the English words with a useful translation to the local language. You must also edit the corresponding expanded messages in the `.MESSAGES.EXPTEXT` directory.

Edit the messages with care. To obtain the support you might require from time to time there must be a one-to-one correspondence between the English messages supplied by the factory and the messages in your language. Support personnel can locate the message associated with each internal error code and can recognize the message number but may not understand the translation. By retaining the internal error code relationship to the message and the message number relationship to explanatory information, you simplify support procedures.

To generate the `.$MSG` and `.$EXPMSG` directories from the files that you edited, execute the batch stream named `.BATCH.BUILD.MESSAGE1` that is shipped with DNOS.



Differences Between DX10 and DNOS

11.1 INTRODUCTION

This section describes some of the differences between DX10 and DNOS that affect migrating from DX10 to DNOS. It also offers advice to help you overcome the differences. The categories of differences are as follows:

- General environment
- Device and file I/O operations
- System Command Interpreter (SCI) user interface
- SCI primitives and interface routines
- Supervisor call (SVC) support
- User-written system software
- System console

11.2 GENERAL ENVIRONMENT

Most DX10 operating system applications programs can execute properly under DNOS. However, the environment in which they run is slightly different. Consequently, the programs might require several subtle changes.

One of the features of DNOS is the job structure which allows the user to isolate a set of tasks with its own resources and run-time environment. Each task in DNOS exists within a job. A job can have job-local logical unit numbers (LUNOs), semaphores, and logical names. Access to many task characteristics is restricted to tasks within a single job. Among other things, the job structure allows some characteristics that are associated with a terminal in DX10 to be independent of a terminal in DNOS.

Some of the particular differences between DX10 and DNOS with respect to the overall environment include the following:

- DX10 has no job architecture; the entire system essentially consists of one job. In DNOS, the job architecture allows users to isolate their tasks and resources from each other. This implies that a number of SCI commands and SVC operations produce different results under the two systems. In particular, status commands such as Show Task Status (STS) and Show I/O Status (SIS) provide status for a particular job name or ID in DNOS rather than displaying status for all tasks as in DX10. Similarly, the Kill Task (KT) command affects only tasks in the user's job in DNOS, while it may be used for any task in DX10. Also, DNOS provides a number of job status commands that are not in DX10. An operator interface is also provided so that a system operator can control any job in the system. The operator job is recognized by the operating system as a privileged job.
- Files written in a DX10 environment may be used directly in a DNOS environment, with the exception of hash-placement key indexed files. Only sequential-placement KIFs are supported in DNOS and in current releases of DX10.
- During a DNOS initial program load (IPL), all disk volumes that are online are installed. In DX10, disk volumes must be explicitly installed.
- DX10 requires a minimum main CPU memory of 256K bytes (where K equals 1024); DNOS requires at least 512K bytes. In general, running DNOS with the same user environment as DX10 requires more main CPU memory.
- DX10 restricts the use of a given user ID and passcode to one terminal at a time. DNOS allows the same user ID and passcode to be active in any number of jobs at any number of terminals at any one time. This feature can have a confusing effect on the permanent synonyms and logical names for the user ID, since each log-on receives the currently defined set of synonyms and logical names from a disk file and each log-off saves the current set to the file.
- The system log device in DX10 can also be used as an SCI terminal. This is not allowed in DNOS, since a terminal being used for SCI is owned and opened by the job using the terminal. When SCI is in use at a terminal, the system job cannot assign a LUNO to the terminal and use it for logging. The log device becomes disabled if the terminal is in use when a log message is to be sent. To establish the terminal as the log device, use the Initialize System Log (ISL) command.
- In addition to the temporary files which can be created with the Create File I/O operation, the DNOS environment supports job temporary files. These files are established using the Assign Logical Name (ALN) SCI command or the Assign Logical Name suboperation of the Name Manager SVC. A job temporary file is available only to tasks within a given job and the file exists only for the duration of the job.

- The hard-break key sequence for DNOS and for DX10 is Attention/(Control)x. The sequence can be redefined by a user who modifies a DNOS table of command definitions. The sequence of tasks killed in response to several break key sequences may differ in DX10 and DNOS. In DX10, only foreground tasks can be killed; SCI, however, cannot be killed. In DNOS, one task is killed each time the break key sequence is used. If only SCI is active, it is killed. The job at the terminal terminates and the terminal is again available for use. If tasks other than SCI are active, the order in which tasks are killed is foreground tasks, then background tasks, then SCI.
- Nonreplicable tasks installed in the program file `.$$PROGA` for DX10 have the same run-time ID as installed ID when they are bid. In DNOS the IDs are not the same when bid from the analogous program file `.$$UTIL`. User-written command procedures that do not check the run-time ID synonym `$$RI` (which is set by the Execute Task (XT) command) must be changed to execute properly in both DX10 and DNOS environments.
- Many internal system structures of DNOS are different from their counterparts in DX10. User programs that access structures such as the task status block (TSB) must be different for the two operating system environments.
- For those with DX10 Release 3.3 or earlier, a Copy Directory (CD) command for a DNOS directory changes channels to aliases and does not copy software privileged tasks in a DNOS program file correctly.
- Names of system files for various uses differ in DX10 and DNOS, as shown in Table 11-1.
- In DNOS, when the destination for installing a task is specified as LUNO 00, file `.$$SHARED` is implied; in DX10, `.$$PROGA` is implied. Specifying attached procedures not from a task's program file implies they are in the `.$$SHARED` program file in DNOS.
- The format of the `S$$CRASH` file differs for the two systems. Therefore, if a disk with `S$$CRASH` written by DNOS is loaded under DX10, it must be loaded twice. The first load fails but prepares the crash file for the second load.
- DX10 provides conversion utilities to transport data in TX990 format to the DX10 disk formats. DNOS does not provide this facility.
- DX5 support is not available with DNOS, but is available with DX10.
- DNOS does not support the TILINK software package, but DX10 does.
- DNOS does not support the 913 terminal and FD800 diskette, but DX10 does.

Table 11-1. DX10/DNOS System File Names

System File Name	DX10	DNOS
Log file 1	.\$\$SLG1	.\$\$LOG1
Log file 2	.\$\$SLG2	.\$\$LOG2
Accounting log file 1	None	.\$\$ACT1
Accounting log file 2	None	.\$\$ACT2
Program file (system kernel)	.\$\$IMAGES	System name
Utilities program file	.\$\$PROGA	.\$\$UTIL
User's shared program file	.\$\$PROGA	.\$\$SHARED
System generation library	.\$\$SYSGEN	.\$\$SGU\$
SCI command library (default)	.\$\$PROC	.\$\$CMDS
Swap File	.\$\$ROLLA	.\$\$ROLLD.\$\$ROLLA

11.3 DEVICE AND FILE I/O OPERATIONS

Because the job orientation in DNOS replaces the station orientation of DX10 to some extent, the scope of LUNOs for I/O is different. DNOS I/O processing in general is more uniform than that of DX10. This introduces some differences in the user interface to the system. File management differs in the variety of options available. DNOS provides a number of options not available in DX10.

Particular differences between DX10 and DNOS include the following:

- In DX10 an Open operation is not required for record-oriented devices. In DNOS an Open operation is required before I/O to any I/O resource is allowed.
- In DX10, the data buffer address field (bytes 6 and 7) of an Assign LUNO I/O SVC block has no data returned to it by the I/O processing. In DNOS, resource type information is returned from the physical device table (PDT). This is different from the data returned by an Open operation. This difference should affect only those programs that reuse an SVC block for another call without clearing relevant fields first.
- In DX10 global LUNO 00 is assigned to ST01; in DNOS it is assigned to DUMMY. This difference should affect only those programs that are not associated with a specific terminal when bid but that perform I/O to LUNO 00.
- Station-local LUNOs, as specified by particular flag settings in DX10 code, are interpreted as job-local LUNOs in DNOS. DNOS does not support station-local LUNOs.
- A task bid by SCI in the DX10 environment can assume that station-local LUNO 00 is assigned to the user's terminal. When a job is created in DNOS, it is given a job-local LUNO 00 assigned to DUMMY. In DNOS, a task-local LUNO 00 is assigned to the user's terminal. This difference should be noticed only by a program which explicitly tries to assign task-local LUNO 00.

- Communications software in DX10 uses the error byte of the I/O SVC block for informative codes as well as error codes. DNOS uses the error byte only for errors and sets the error flag in the system flags byte whenever the error byte is nonzero. DX10 communications software does not expect the error flag to be set for informative codes.
- A task that enters its end-action routine in DX10 has LUNOs released before entering the end-action code. DNOS tasks can use the LUNOs in their end action code without having to reassign them because DNOS does not release LUNOs before taking end action. DNOS time-out during end action can be set using the Modify Scheduler/Swap Parameter (MSP) command. The default is 100 system time units.
- DNOS and current releases of DX10 support sequential-placement KIFs. Hash-placement files created under previous versions of DX10 must be converted to sequential-placement files on a DX10 system before attempting to use the files with DNOS.
- On any KIF I/O operation in which a key is specified, DNOS requires that the key specified flag be set in the user flags byte (byte 5) of the SVC block, while DX10 does not. The operations for which the flag must be set include the following:
 - Subopcode 41 (Read Greater)
 - Subopcode 44 (Read Equal or Greater)
- The extended call block flag of the user flags byte (byte 5) of the I/O SVC block is used in TX990 to mean logical track addressing. DX10 contains a special code to ignore it if set for devices that are not terminals. DNOS does not contain this code.
- The DNOS I/O system stores two more words of the I/O SVC block than DX10 does for sequential file I/O. These two words are not returned to the task. Therefore, a problem arises only if the call block is at the end of the task address space where two more words are not available.
- The device LP\$1 designates use of LP01 as a print device in DX10. In DNOS, LP\$1 functions as a print device only if it is defined as a logical name for one of the line printers in the system.

11.4 SCI USER INTERFACE

DNOS SCI provides essentially the same user interface to the operating system as DX10 SCI. Several differences take advantage of new features in DNOS, and some differences provide a more uniform interface. A number of enhancements in speed and space have been developed. The specific differences are as follows:

- DNOS provides a synonym and logical name table for each job; DX10 has a synonym table for each station. The tables for DNOS are considerably larger than those for DX10.
- After a DNOS system is generated, a Modify Spooler Device (MSD) command must be executed for each device to be used with the Print File (PF) command. MSD sets up the appropriate tables for the spooling of output. All output from PF is through the spooler subsystem of DNOS. When the spooler is active, all devices that have been modified to be spool devices are available only to the spooler. Therefore, a command such as List Directory (LD) cannot have its output directed to a spooled device. It can be directed to a file which is then printed with PF, to a nonspooled device, or to a spooler logical name (that is, a logical name that has been assigned to a device available to the spooler). Batch streams written for DX10 that specify a line printer for output listings may require modification to run on DNOS.
- The Print File (PF) command in DX10 calls the Show Output Status (SOS) command to show the current queue of files to the device specified for output. The DNOS PF command does not call SOS; the user must explicitly specify SOS after PF to monitor the output.
- In DX10 the status of each terminal is reinitialized via a Modify Terminal Status (MTS) command during each IPL. In DNOS this is not required because the MTS command modifies the status on a disk file and not in memory only. This may not be a problem for most users. However, it may be a problem to the user who expects the terminal to return to the default status during an IPL.
- DX10 SCI supports two types of task modes, foreground and background. DNOS supports the same two types of task modes in interactive jobs. In addition, DNOS supports batch jobs in which tasks run independently of a terminal. Batch jobs, like background tasks in interactive jobs, request that SCI process a file of commands (batch stream) as its primary input. Some batch streams that can be executed with Execute Batch (XB) in DX10 or DNOS cannot be executed with Execute Batch Job (XBJ) in DNOS. This is because the values of the synonyms \$\$ST and ME and the pathname of the terminal local file (TLF) are different when SCI is started with the XBJ command. The value of \$\$\$ST is 00, and the synonym ME is not assigned when using XBJ. This problem should not occur for most batch streams.
- The primary command procedure library is named .\$\$CMDS in DNOS; it is .\$\$PROC in DX10. Batch streams that include a .USE that specifies .\$\$PROC for DX10 must specify .\$\$CMDS for DNOS.
- In DX10 each of the major subsystems reports error and status conditions in its own way; in DNOS the error reporting mechanism is the same for all subsystems.

11.5 SCI PRIMITIVES AND INTERFACE ROUTINES

The structure of the SCI task in DNOS is different from that in DX10. DX10 SCI consists of a task with a number of overlays for various support functions such as modifying synonyms, handling user IDs, editing, and debugging. In DNOS, SCI is a single task and most of the utility functions are performed by independent tasks. Other differences in SCI interfaces and the primitives include the following:

- User-written SCI command procedures and batch streams that directly bid standard SCI utilities using `.OVLY` or `.BID` instead of using the standard command procedures in DX10 do not execute without modification for DNOS. The reasons are as follows:
 - The parameters for `.BID` have been augmented and defaults have been changed. Specifying a bid from `LUNO = 00` indicates use of the program file `.$$PROGA` (the system program file) in DX10, but it specifies `.$$SHARED` (the shared procedure program file) in DNOS. The program file in use for `.BID` can be specified explicitly using the `PROGRAM FILE =` option in DNOS. Use of `UTILITY` as a keyword specifies a bid from the utilities program file in DNOS, `.$$UTIL`.
 - `.OVLY` is not supported in DNOS.
 - Task IDs for standard system utilities in DX10 are not the same as those used for DNOS. Therefore, bidding a task using the same installed ID for both operating systems probably fails.
 - Some utilities require different `PARMS` lists on the `.BID` for DNOS and for DX10.
 - Some utilities write out headers in DNOS using the `.DATA` primitive in the command procedure rather than having the text for the header in the utility itself, as in DX10. The effect of this difference is that in some cases of errors, the header appears as well as the error and a carriage return must be performed to receive the error message.
- The synonyms representing Text Editor active (`$$EA`) and Debugger active (`$$DA`) are set by different tasks and to different values in DNOS than in DX10. In DX10 these synonyms are set by SCI just before reading a command from its primary input device (terminal or batch stream). In DNOS these synonyms are set by the Text Editor and Debugger tasks. In addition, these synonyms are both set by DX10 SCI to a value of `N` when SCI is started. DNOS SCI does not do this. The synonyms have no values until the appropriate utility sets them. Any user-written command procedure that accesses these synonyms may require changes.
- Any user task that is linked with SCI run-time routines for DX10 must be relinked with DNOS SCI run-time routines. The DNOS versions of `S$` routines must be used for tasks that run in the DNOS environment; the DX10 versions do not run in DNOS. The same caution applies to linking language programs with the `S$` routines.
- If an error occurs on the open of a file in `S$OPEN`, the error code returned is `> 906A` and not `> 01xx`, where `xx` is the error byte.

- Programs can be written in DX10 that link in S\$STOP without linking in S\$MAPS and S\$SETS. This is not possible in DNOS. Two problems might arise:
 - An error can occur when linking the program if the link control file explicitly includes the S\$ routines rather than making use of a LIBRARY command at the beginning.
 - The program can require more memory. This is not likely, however, since most of the DNOS S\$ routines are shorter than the DX10 versions.
- In DNOS, the synonym \$\$CC is set by some standard utilities that do not set the synonym in DX10. Many utilities in DNOS set \$\$CC to a different value. This change was made to support DNOS error handling facilities. The change should not be noticed unless a batch stream includes a test for an explicit value of \$\$CC instead of testing for zero or nonzero values. Batch streams that must check for a particular error code in DNOS must access the synonym \$\$MN instead of \$\$CC. See the *DNOS Messages and Codes Reference Manual* for the exact message number to match a particular error code.
- User IDs in DNOS can be eight characters long but only six characters long in DX10. Therefore, the value of the synonym \$\$UI can be set by SCI to an eight-character value in DNOS and a six-character value in DX10. This may be a problem when the synonym value is used as part of a pathname and consists of eight characters in DNOS.
- The left bracket, right bracket, and back slash characters are valid characters in items of type NAME in DNOS SCI. For example, .OPTION PROMPT=[@ME] results in the prompt of [ME] in DNOS rather than the station number enclosed in brackets as in DX10. These characters serve as terminators when DX10 SCI scans for the end of an item of type NAME. Internationalization requires this change because several other alphabets require that equivalent ASCII code be allowed in items of type NAME. Using [@ME^] resolves to the station number in either system.

11.6 SVC SUPPORT

The following are the differences between DNOS and DX10 support of SVCs in user tasks:

- In DX10, a task that issues an undefined SVC is terminated or goes into its end-action routine. DNOS returns an error code in the SVC block. The task continues to execute at the instruction following the SVC.
- DNOS supports the following SVCs that DX10 does not support:
 - SVC > 00: Create IPC Channel (subopcode 9D), Delete IPC Channel (subopcode 9E), Master Read (subopcode 19), Read Call Block (subopcode 1A), Master Write (subopcode 1B), and Redirect Assign LUNO (subopcode 1C) for interprocess communication (IPC).
 - SVC > 3D (Semaphore Operations)
 - SVC > 40 (Segment Management)

- SVC > 41 (Initiate Event)
- SVC > 42 (Wait for Event)
- SVC > 43 (Name Management)
- SVC > 45 (Get Encrypted Value)
- SVC > 46 (Get Decrypted Value)
- SVC > 47 (Log Accounting Entry)
- SVC > 48 (Job Management)
- SVC > 49 (Get Accounting Information from TSB)
- SVC > 4A (Modify BTA or JCA Size)
- SVC > 4C (Return Code Processor)
- SVC > 4F (Post Event)
- In DX10, a task can issue the following SVCs to affect any other task; in DNOS, these SVCs affect only tasks within the requesting task's job:
 - SVC > 07 (Activate Suspended Task)
 - SVC > 0E (Activate Time-Delayed Task)
 - SVC > 2C (Read/Write TSB)
 - SVC > 2D (Read/Write Task)
 - SVC > 33 (Kill Task)
 - SVC > 35 (Poll Status of Task)
- The following SVCs have options in DNOS that are not available in DX10. Each of these uses fields that are marked as reserved in DX10. If DX10 tasks using these SVCs have all reserved fields set to zero, no compatibility problems should occur.
 - SVC > 1F (Scheduled Bid Task)
 - SVC > 26 (Install Procedure Program Segment)
 - SVC > 2B (Bid Task)
 - SVC > 2E (Self Identification)

- In DNOS, SVC > 0F (Abort I/O Requests by LUNO), has a different call block format and a different scope than in DX10. The SVC is valid only for privileged tasks in DX10. Any task may issue the SVC for its own LUNOs in DNOS.
- In DNOS, SVC > 3F (Retrieve System Data) has a different call block format than in DX10 to provide access to the new data structures of DNOS.
- In DX10, it was recommended that users identify messages passed using the intertask message queue by the task run ID of the task to receive the message. In DNOS, task run IDs are job-local rather than global. Thus, users of the Get Data (> 1D) and Put Data (> 1C) SVCs need to be aware of this difference. DNOS users should use IPC for this type of exchange and avoid SVCs > 1C and > 1D.
- In DX10, the table area reserved for the intertask message queue is specified during system generation. In DNOS, the size of the queue is set to be a maximum of 2,000 bytes.
- The following SVCs supported by early releases of DX10 are not supported by DNOS. In some cases, features of DNOS replace the functions of the SVCs no longer supported.
 - SVC > 00, subpcodes > 8x (not supported since DX10 2.2)
 - SVC > 05 (Bid Task; replaced by > 2B)
 - SVC > 08 (Unconditional Character Input)
 - SVC > 15 (Disk Utility; not supported since DX10 2.2)
 - SVC > 16 (End of Program; replaced by > 04)
 - SVC > 18 (Conditional Character Input)
 - SVC > 19 (Set Condition Bit; not supported since DX10 2.2)
 - SVC > 1A (913 VDT Utility; not supported since DX10 2.2)
 - SVC > 1E (Abort I/O by Call Block Address; not supported since DX10 3.2)
 - SVC > 23 (Make Task Privileged)
 - SVC > 30 (Get Event Character)
 - SVC > 32 (Get System Table Address; not relevant to DNOS)
 - SVC > 39 (Get Event Character by LUNO)
 - SVC > 3A (Set Event Key; not supported in DX10)
 - SVC > 3C (Diagnostic Dump; not supported in DX10 systems)

- Some DNOS SVC processors return error codes that DX10 does not return. Some of the DX10 codes represent errors that cannot occur in DNOS; these codes have been assigned new meanings. If a program executes in both operating system environments and checks for a particular error code or for all error codes, the program must be reviewed carefully to ensure it is correctly checking for errors. A determined effort was made to retain identical codes in DX10 and DNOS for many common conditions.
- The Read/Write Task SVC (> 2D) processes requests to read or write to an odd-byte address boundary in DNOS. In DX10 the address is rounded down to the nearest even address.

11.7 USER-WRITTEN SYSTEM SOFTWARE

Since the internal data structures of DNOS differ from those of DX10, any user-defined SVC processors that must be used with both systems and that access system data structures must have source code modifications. User-defined SVC processors that are independent of the operating system structures need minimal modifications from DX10 format to fit into the DNOS table-driven scheme of access. All user-defined SVC processors must be reassembled and included in system generation to be linked into DNOS. Section 4 describes requirements for user-defined SVCs.

User-written DSRs that must be used with both DX10 and DNOS must be rewritten for DNOS to make use of new interfaces to the I/O subsystem and to conform to changes in data structures. Consult the *DNOS System Design Document* and the section of this manual concerning DSRs for further information.

Other user-written system tasks that access system data structures and are used with both operating systems must be modified for use in DNOS. See the *DNOS System Design Document* for details on system internals and how to write system tasks.

11.8 SYSTEM CONSOLE

DNOS allows a terminal to be designated as the system console. The user of that terminal becomes the system operator. DX10 has no parallel function in DX10. Some commands that have a global scope in DX10 have a job scope for DNOS users and a global scope for the DNOS system operator.

To designate a terminal as system console and the user who is logged on at that terminal as system operator, enter the Execute Operator Interface (XOI) command.

The command fails if a system console designation is in effect for another terminal. If no terminal is designated as system console, the terminal at which the command is entered becomes the system console and continues to be the system console until a Quit Operator Interface (QOI) command is successfully executed at the system console. Refer to the *DNOS System Command Interpreter (SCI) Reference Manual* for further information about these commands and to the *DNOS Operations Guide* for general instructions on using the operator interface.

A user must be designated system operator in order to perform the following functions:

- Show status of any job in the system
- Show status of all I/O in progress in the system
- Force abnormal termination of any task in the system

The system operator receives operator messages and requests until pressing the CMD key to activate SCI and enter other commands. Any messages are queued until the system operator again enters the XOI command.

Operator messages are displayed following execution of each SCI command if the system operator enters a Receive Operator Messages (ROM) command while SCI is active. The command is as follows:

```
[ ] ROM
RECEIVE OPERATOR MESSAGES
MESSAGES: ALL
```

The system operator should enter the XOI command again after entering the SCI commands (that is, when the terminal is idle). Otherwise, messages are not displayed.

When an operator message contains a request ID followed by an asterisk, the system operator must respond. The task that issued the request is suspended pending receipt of the response. When the system operator has performed the requested function and the operator interface task is active, the operator presses the F4 key and enters the number of the request to which he is responding. If the system operator is unable to perform the request and the operator interface task is active, the operator presses the F5 key, enters the number of the request, and indicates whether or not to terminate the request. The operator uses the Respond to Operator Interface Request (ROR) command and the Kill Operator Interface Request (KOR) command (which is described in the *DNOS System Command Interpreter (SCI) Reference Manual*) instead of the F4 and F5 keys when SCI is active.

Special Features of DNOS

12.1 INTRODUCTION

You can use a number of features in DNOS to alleviate problems that exist in the DX10 environment. Many of these enhancements have no counterparts in DX10. The following paragraphs describe ways to utilize these DNOS features. Further detail is available in other DNOS manuals.

12.2 DNOS JOB ARCHITECTURE

DX10 users often require several background tasks to be active at the same time. As a DNOS user, you can activate batch jobs with the Execute Batch Job (XBJ) command. You can initiate any number of batch jobs at one time since each of these jobs is independent of any terminal and of any other job. However, during system generation, the systems programmer specifies the limit to the number of jobs that can be active at one time; the default value is 32. Any jobs initiated after this limit is reached are placed in a queue. When an active job terminates, a job from the queue becomes active. In addition to the active job limit, there is also a batch job limit. The systems programmer can therefore govern not only how many jobs are active, but how many can run in batch mode at the same time.

Several users may be logged on with the same ID in different jobs or the same job. When several users are logged on at different terminals, but in the same job, they can each have their own SCI task governing the interaction. In this situation, each user begins the interactive session with the set of synonyms and logical names currently saved on disk as the permanent copy for that user ID. When the last user quits, his current set of synonyms and logical names becomes the new permanent copy. Having several users in the same job requires less system table area for job structures than having each user run in his own job.

A feature known as the keyboard bid is used when you log on to SCI. By pressing the attention key followed by the exclamation point, you bid a log-on task, which in turn bids SCI. You have control of the task that is bid for this sequence. During system configuration, you can provide a table of entries showing what task to bid for a given sequence of attention plus some other key. You can define the sequence to bid a task after bidding a log-on task, or you can define it to bid a task within a running job. For example, you might define your own task to implement the PRINT key or to replace SCI. Details about using this feature can be found in the I/O section of the *DNOS System Design Document*. The SCI commands used with this feature are the Modify Command Definition Table (MCDT), Show Command Definition Table (SCDT), and Modify Device Configuration (MDC) commands. These commands are described in the *DNOS System Command Interpreter (SCI) Reference Manual*.

12.3 LOGICAL NAMES

Logical names provide several features in DNOS including concatenated files, multifile key indexed file (KIF) sets, and access to the spooler from user tasks.

You can create a concatenated set of sequential files or a concatenated set of relative record files by using the Assign Logical Name (ALN) command. Subsequently, the concatenated set is a single file. Access to the logical name from the SCI command level or from the user task actually affects the appropriate physical file of the concatenated set. The files can reside on one or more volumes.

You can also use the ALN command to create a multifile KIF set. By using a logical name, you can treat two or more KIFs having the same characteristics as one file. The files in use can reside on one or more volumes. When you execute the ALN command, all files specified after the first file must be empty. This feature can extend a KIF that nearly fills a volume or uses as much space as desired on a single volume. When the first file becomes full, a second is created on another volume using the same characteristics as the first file. You then use the ALN command to specify a logical name associating the two files.

You can also use logical names to customize user access to the spooler. Once you have made a device available to the spooler by using the Modify Spooler Device (MSD) command, users can direct output to that device by using the Print File (PF) command. You can also define your own logical names for particular spooler devices by using the ALN command. When you specify a spooler logical name, you can provide the default parameters for that name including such items as number of copies to print, whether or not to delete the file after printing, and the number of lines per page. You can also assign a global logical name to access spooler devices that are available to all system users.

Once a Spooler device has an assigned logical name, you can use that logical name in any SCI command that generates an output listing. For example, you might respond to the LISTING ACCESS NAME prompt for the Execute Macro Assembler (XMA) SCI command with OUTPUT, if OUTPUT is a logical name for LP01. You can also use the PF command with the selected logical name. In addition, you can specify the logical name as your output device or file name for output. In all cases, the Spooler queues the output file for printing at the device specified with the ALN command.

12.4 SCI FEATURES

A number of SCI features are obvious to users of DX10; others are more subtle. The following list describes the less obvious features:

- Error messages are of a uniform format and style for utilities supported on DNOS.
- A synonym and logical name table is provided. Each table has as many as 12,000 bytes of space.
- A fast VDT mode is available when the VDT terminal is identified as DEFAULT MODE=VDT in the Modify Terminal Status (MTS) command. This mode has several advantages:
 - Menus are displayed quickly, using a single operation.
 - Each screen of data displayed by the Show File (SF) command is written with a single operation.
 - Function keys F6 (line numbers), F3 (horizontal scroll left), F4 (horizontal scroll right), Next Field, and Previous Field are effective with the SF command.
- Graphics characters can appear in menus.
- The Enter key allows you to accept all of the current prompt responses to a command procedure shown on the screen.
- The Erase Input key allows you to reset any initial values of the responses shown on the screen and return to the first command prompt shown on the screen.

12.5 INTERPROCESS COMMUNICATION (IPC)

DX10 tasks use several limited mechanisms to exchange information between tasks. These include access to the system common data area and use of the intertask message queue with the Get Data SVC (> 1D) and Put Data SVC (> 1C). DNOS IPC provides a considerably more flexible tool for communication. IPC functions like an I/O resource, making use of the I/O SVC (> 00) as the transmission mechanism.

DNOS supports IPC flexibly from assembly language programs. The high-level languages and applications environment processors use IPC internally. High-level languages can use symmetric channels for resource-independent I/O like any I/O device, making use of ASCII Read and ASCII Write operations. Use of master/slave channels, however, requires careful writing of the owner task by using SVCs.

12.6 PERFORMANCE MONITORING

The following utilities are available with both DX10 and DNOS for monitoring performance. These include the following:

- The memory map available from the Show Memory Map (SMM) command
- The memory statistics from the Show Memory Status (SMS) command
- The current set of tasks active (in a job) as shown by the Show Task Status (STS) command
- The various terminal, LUNO, and output status lists

In addition, DNOS provides the following monitoring utilities:

- Current table use and activity levels for various DNOS subsystems from the Execute Performance Display (XPD) command
- The current set of jobs (and, optionally, tasks in these jobs) from the Show Job Status (SJS) command and the List Jobs (LJ) command
- A dynamic display of tasks in a given job from the Execute Job Monitor (XJM) command

12.7 PERFORMANCE OPTIMIZATION

You can use several features of DNOS to optimize the performance of the 990 minicomputer. The following features aid in extending the device capacity and execution performance of the computer:

- Alternate ways to structure SCI sessions
- Keyboard bid of application tasks
- Types of LUNOs
- Batch jobs
- Miscellaneous items

12.7.1 Alternate Ways to Structure SCI Sessions

DNOS is flexible in the way in which SCI sessions are structured and started. The following features are useful in structuring an SCI session for your specific needs.

- SCI is available to users in their own jobs or in the same jobs as other users.
- You can log on to SCI from one terminal and then, by using the Execute Task (XT) command, activate other terminals.
- You can log on to SCI at one terminal and start a user task that bids tasks at other terminals.

Each execution of SCI uses approximately 100 bytes of system table area. You might want to use less system table area for applications that do not require SCI. However, applications developed with DNOS need access to synonyms and logical names that are created in the SCI environment. To access these, you must start the job executing without SCI in an environment that is defined by using SCI. You have several options for providing that environment.

You can use an SCI job to establish a set of synonyms and logical names and save these in a file. When you log on using that ID, those synonyms and logical names are available. Terminals modified by the Modify Terminal Status (MTS) command can request name management files. By using the Snapshot Name Definitions (SND) command, you can save the names and parameters for the task that will be the application task to execute without SCI. You can then start the application task by using the keyboard bid. This allows applications that are written in high-level languages to execute without SCI.

You can also use the SND command to save the required names for the application environment but not specify the parameters. A keyboard bid can then start an assembly language task that uses the S\$BIDT routine from the SCI S\$ routines to bid a task written in a high-level language and supply it with the required parameters.

12.7.2 Keyboard Bidding of Tasks

A DNOS user can bid an application task by using a predefined keyboard sequence. A keyboard bid sequence can be defined as active for some terminals but inactive for others. Keyboard bids are especially useful in environments where SCI is used only to bid an initial program. Direct initiation of the application would eliminate the need for SCI entirely and reduce the amount of system overhead. Keyboard bids are processed by a Device Service Routine (DSR). For more details on the internal processing, refer to the section entitled Writing a DSR.

CAUTION

DNOS currently has keyboard bid sequences defined. Texas Instruments reserves the right to add sequences during subsequent releases of DNOS.

Keyboard bidding makes use of the system directory `.$CDT`. A file is created in this directory the first time an initial program load (IPL) takes place. The name of the file will be identical to the name of the kernel program file in use. This file contains a record for each device type. These records are known as Command Definition Tables (CDTs). One CDT pertains to 911s, another for 931s and so on. Each CDT may contain up to 16 command definition entries (CDEs). If you want more than one terminal type to have keyboard bid access to a task, you will need to modify the CDT for each terminal type.

A CDE contains the information needed to execute a keyboard bid from the associated device. The keyboard sequence you use must have the following format: Attention key followed by another keyboard character. The selected character is stored in the CDE, along with the task ID and global LUNO for the program file from which the task is to be bid. The CDE may also contain two words of task bid parameters.

Several options are indicated by a CDE. One may specify if the task is to be bid in the current job or in a new job. If the task is to be bid in a new job, a logon task is needed to create the job. The logon task ID and program file LUNO are stored in the CDE. If a logon task is used, an option allows the station number and/or the bid character as task bid parameters to be passed to the logon task.

If the DNOS logon task is used to bid the application task, the even loading option is supported. With even loading, the task is bid in an existing job. The user ID of the job can be prompted for or specified in the CDE. All jobs running under the specified user ID are searched; the task is bid in the job containing the fewest tasks.

The SCI commands provided to set up the command definition tables include the following:

1. The Show Command Definition Table (SCDT) command allows you to examine the defined command definition entries for a given device in the command definition table.
2. The Modify Command Definition Table (MCDT) command allows you to add or delete command definition entries to a command definition table.

SCDT and MCDT are used to define keyboard sequences. The next step is to enable or disable sequences for specific terminals.

During system generation, a CDE mask is defined for each device. The bits in this mask correspond to an entry in the CDT associated with the device. For example, the CDT for a 931 might have the first four entries defined, indicated by a mask of `> F000`. For a particular terminal, this mask may be changed by using the Modify Device Configuration (MDC) command. Each terminal which uses something other than the supplied default sequences must have its CDE mask changed. Changes made will take effect after the next IPL.

Refer to the *DNOS System Command Interpreter (SCI) Reference Manual* for details concerning the SCDT, MCDT, and MDC commands.

12.7.3 Types of LUNOs

When you assign global LUNOs to files, some structures are placed in the system table area. If you wish to minimize the use of system table area, avoid using global LUNOs.

While structures for global LUNOs are placed in the system table area, structures for job-local and task-local LUNOs are placed in the user's job communication area (JCA). These structures include the logical device table (LDT) and resource privilege block (RPB) for any assigned LUNO. An LDT is 20 bytes long and an RPB is 18 bytes long. Other structures may also be in the JCA, if the LUNO is assigned to a file or IPC channel.

If a job has several tasks and each of these tasks accesses many resources, the large number of associated LUNO structures may tax the JCA. In this case, you may wish to consider shared LUNOs. The shared LUNO capability of DNOS minimizes the number of such structures for the same resource for tasks in the same job. When a given task assigns a shared LUNO to some resource and opens that LUNO, other tasks in the same job can also use the same LUNO without an Assign operation. Rather than having multiple LDTs and RPBs (that is, one pair for each Open operation issued), you have one LDT and as many RPBs as Assign operations.

More than one task at a time can open a shared LUNO, but only one task at a time can open a job-local LUNO. Each task that uses a shared LUNO must open that LUNO and is subject to the standard cross checking with other Open operations.

No LUNOs are assigned to the program file when tasks are bid from .S\$UTIL or .S\$SHARED with system-supplied commands. However, a LUNO is assigned when the .BID is from SCI, and that LUNO remains until the task that was bid terminates.

12.7.4 Batch Jobs

DNOS can process both batch jobs and interactive jobs. Use of the Execute Batch Job (XBJ) command allows you to execute batch streams independent of terminals, leaving those terminals free for other operations.

12.7.5 Miscellaneous Items

A Copy Directory (CD) command of the system disk to a clean disk eliminates fragmentation (secondary allocations). This aids in minimizing secondary allocation table (SAT) structures for any shared files with structures in the system table area.

Since a six-byte structure is built in the system table area for each directory level of a file name, complex directory structures can use large amounts of the system table area. Consequently, when organizing a disk directory structure, you should use a minimum number of directory levels for files that you frequently access. However, you should use at least one directory level and avoid putting files directly under .VCATALOG. Any change to a file directly under .VCATALOG locks .VCATALOG during an update and slows system response. Also, your directory should be large enough so that it is never more than 90 percent full. The 10 percent unused space maximizes the hashing algorithm.

12.8 SYSTEM CONFIGURATION UTILITY

Modifications to the system table sizes and supported devices of DX10 require use of the system generation utility. In DNOS, you can make changes to tables, devices, and a number of other characteristics using an interactive utility while the system is running. The utility is accessed by the Execute System Configuration Utility (XSCU) command. Some of the changes are made while executing the utility. Others take effect when the system goes through its next IPL.

12.9 \$\$SYSTEM DIRECTORY

DNOS provides several tools for both the systems programmer and Texas Instruments field personnel in the directory .\$\$SYSTEM on the DNOS system disk. This directory includes the file .\$\$SYSTEM.\$\$HSTRY and the command procedure directory .\$\$SYSTEM.\$\$\$CMDS.

The \$\$HSTRY file records information concerning the installation of software products sold by Texas Instruments. It is updated by the installation batch streams for languages and applications environment processors, by the Modify Volume Information (MVI) command, and by the commands used to install a new version of DNOS. This history file is interrogated by the List Software Configuration (LSC) command when producing a report of the current state of software on the system.

The command procedure directory .\$\$CMDS includes the procedures needed to build the history file. Users who develop large applications might be interested in maintaining history records about installation of those applications. It also includes the XPROCCVT command to aid in converting DX10 batch streams and command procedures for DNOS. Other members of the .\$\$SYSTEM.\$\$\$CMDS directory are not intended for use by users.

12.10 SYSTEM COMMAND PROCEDURES

DNOS includes general system command procedures, which are described in the following paragraphs.

12.10.1 Begin Update Documentation (BUD)

When used in a batch stream, the BUD command starts an update documentation sequence that ends with an EUD (End Update Documentation) command. It sends a message to the history file (.\$\$SYSTEM.\$\$HSTRY) indicating that installation or patching of the specified software product has begun. When used interactively, BUD does both the update and end update operations.

Prompts

```
BEGIN UPDATE DOCUMENTATION
```

```

VOLUME BEING UPDATED: volume name
  PACKAGE NAME: character(s)
  RELEASE NUMBER: character(s)
  RELEASE DATE: character(s)
SYSTEM COMPONENT NAME: character(s)
  BEGIN NEW ENTRY?: [YES/NO]           (YES)
```

*Prompt Details***VOLUME BEING UPDATED**

The disk volume for which the history information for this installation or patch batch stream is being executed.

PACKAGE NAME

From 1 to 33 characters specifying the product name or description. It may also be a part of a product, as in the case of products with pieces that are maintained or patched as a separate entity (for example, the kernel and utility portions of the DNOS operating system).

RELEASE NUMBER

From 1 to 7 characters representing the release level or revision level of the associated software product.

RELEASE DATE

The eight-character date entered in the format MM/DD/YY and associated with the release number.

SYSTEM COMPONENT NAME

From 1 to 11 characters specifying the keyword unique to each software product. This is essentially an abbreviation for the package name. Keywords beginning with DN and UT are reserved for the DNOS operating system. Other keywords are used by various products. Texas Instruments may modify or add to these keywords as new products become available without regard to conflict with any user's use of this command.

BEGIN NEW ENTRY?

YES or NO. This prompt controls formatting of the history file (.\$\$SYSTEM.\$\$HSTRY). You should accept YES (the initial value) except for DNOS kernel patching. For kernel patching, enter NO.

Examples

```
BUD VOLUME BEING UPDATED=DS01,
    PACKAGE NAME=DNOS OPERATING SYSTEM (KERNEL),
    RELEASE NUMBER=1.0.0, RELEASE DATE=8/01/81,
    SYSTEM COMPONENT NAME=DP, BEGIN NEW ENTRY?=NO
```

```
BUD VOLUME BEING UPDATED=DS01,
    PACKAGE NAME=DNOS OPERATING SYSTEM (UTILITY),
    RELEASE NUMBER=1.0.0, RELEASE DATE=8/01/81,
    SYSTEM COMPONENT NAME=UT.$$UTIL, BEGIN NEW ENTRY?=YES
```

Assumptions

A subsequent EUD command will be executed.

Notes

1. The command is located in an alternate command directory reserved for Texas Instruments commands. You must execute the primitive `.USE.S$$SYSTEM.S$$CMDS` to access this command.
2. The synonym `$OT` controls the target volume on which the `.S$$SYSTEM.S$$HSTRY` file resides.
3. Since the BUD procedure is composed of SCI primitives only, it does not set the `$$CC` synonym and should not be followed by an EC command. If an error occurs in the BUD command, the batch stream in which it is used terminates.

Related Commands

EUD End Update Documentation
LSC List Software Configuration

12.10.2 End Update Documentation (EUD)

The EUD command terminates an update documentation sequence that a BUD command began. It sends a message to the history file (`.S$$SYSTEM.S$$HSTRY`) indicating that installation or patching has completed. It also indicates the number of the last patch applied.

Prompts

```
END UPDATE DOCUMENTATION

      LAST PATCH: [<integer>]
RELEASE NUMBER: [character(s)]
```

Prompt Details

LAST PATCH

A user-defined integer value. This value allows you to monitor the last patch applied. For an installation batch stream, this field must be null in order to update the history file correctly.

RELEASE NUMBER

From 5 to 7 characters representing the release level or revision level of the associated software product. The format is `v.r.e`, where `v` represents the version of the product, `r` represents the revision level, and `e` represents the latest change. This format must be followed exactly for the List Software Configuration (LSC) command to work properly.

Assumptions

A prior BUD command is executed.

Notes

1. This command is valid only in batch streams and is intended for use by Texas Instruments supplied software package installation and patch batch streams. The command is located in an alternate command directory reserved for Texas Instruments commands. You must execute the primitive `.USE .$$SYSTEM.$$CMDS` to access this command.
2. Since the EUD procedure is composed of SCI primitives only, it does not set the `$$CC` synonym and should not be followed by an EC command. If an error occurs in the EUD command, the batch stream in which it is used terminates.

Related Commands

BUD Begin Update Documentation

12.11 MAINTAINING USER IDS

If you attempt to copy a set of user IDs from one system disk to another, you must copy both the `.$$USER` directory and the `.$$CLF` file. If these two items are not from the same system version, various types of crashes and log-on problems can occur. Similarly, if you create a new system on an old system disk, you can preserve the old user IDs only if you preserve both the `.$$USER` directory and the `.$$CLF` file.

When changing from DNOS 1.1 to DNOS 1.2, the old user IDs cannot be used because of changes needed to support file security on DNOS. You must assign new user IDs for the DNOS 1.2 system.

12.12 SPOOLER SUBSYSTEM

The spooler subsystem is the interface between users and output. It is designed to prevent unauthorized deletion or modification of output requests, yet give you complete control of requests that you initiate. The spooler subsystem consists of tasks that execute in the spooler job and tasks that execute in the user's job. The spooler job is created by the initialization batch stream, `.$$ISBTCH`, which contains an Execute Job (XJ) SCI command.

12.12.1 Spooler Directory

The spooler directory, `.$$SDTQUE`, must reside on the system disk. This directory contains two types of files, a banner sheet file and spooler queue files.

12.12.1.1 Spooler Banner Sheet File. The banner sheet file, `.$$SDTQUE.$$BANNER`, specifies the format of the banner sheet that is displayed if you specify YES in response to the BANNER SHEET prompt in the Print File (PF) SCI command or in the \$\$SPLR routine. The standard banner sheet displays the user's job name, the user ID, the pathname of the file being printed, the time, and the date. You can edit the banner sheet file to display customized banner sheets. The banner sheet file consists of various kinds of records that are 80 characters long. The following records are command records, which indicates that they are special display requirements:

Command Records	Description
<code>/JOB</code>	Displays enlarged user's job name
<code>/USER</code>	Displays enlarged users ID
<code>/DATE</code>	Displays current date and time
<code>/TEXT,CCCCCCCC</code>	Displays enlarged characters CCCCCCCC
<code>/FILE</code>	Displays requested print file name

If a record in the banner file is not a command record, the output device displays that record. By using the `/TEXT` command record and by editing the banner sheet file to contain specific display lines, you can create your own customized banner sheets.

12.12.1.2 Spooler Queue File. The spooler queue file does not exist before the first initialization of your DNOS system. The spooler automatically creates the file. The name of the spooler queue file is `.$$SDTQUE.< name of the generated operating system >`. Each operating system that is generated has a unique queue file. This avoids possible conflicts in hardware differences that each generated operating system supports.

The spooler obtains the task bid parameters specified on the SCI Execute Job (XJ) command in the system initialization batch stream. These two items, PARM1 and PARM2, allow each installation to configure the spooler queue file to its particular needs. The structure of the queue file is described in the *DNOS SCI and Utilities Design Document*.

The queue file contains a number of class name records. These records contain class name entries that are pseudo names associated with specified devices. The PARM1 parameter in the XJ command specifies the number of class name records that are to be included in the file when it is created. Each class name record can contain 48 class name entries; the class name table (CNT) template describes the class name record.

The queue file contains a number of device table records. The device table records contain device entries. These entries contain information about each device available to the spooler or known to the spooler. The PARM2 parameter specifies the number of device table records that are to be included in the spooler queue file when it is created. Each device table record can contain 12 device entries; the spooler device table (SDT) template describes the device table record.

By editing the initialization batch stream `.$$ISBTCH`, you can structure the queue file for the particular needs of your installation. For example, if one class name record is sufficient (that is, no more than 48 class names are required) but 30 spooler devices are required, specify the XJ PARM1 value as 1 to obtain the 48 class names. To obtain three device table records, specify The XJ PARM2 value as 3. Each device table record contains 12 entries, which support a total of 36 devices.

The first record of the queue file contains the name of the file, the version number of the operating system, the number of class name records, and the number of device table records. If the file exists already, the name in the file is compared to the name of the generated operating system. The version in the file is compared to the version of the operating system. The number of queue file class name records is compared to the value specified by PARM1, and the number of queue file device table records is compared to the value specified by PARM2. If any of these items do not correspond, the current file is deleted and recreated using the current parameters.

Each record of the spooler queue file requires 768 bytes. The minimum size of the spooler queue file is four records, and the maximum size is 65,535 records. The file expands as it needs more space to continue operations. It is recommended that you regularly examine the size of the queue files in the .SSDTQUE directory. You should delete old queue files that correspond to unused operating systems. If the current queue file is too large and is not needed, then you should follow the Spooler Clean-up routine described in paragraph that follows in this section.

12.12.2 Spooler Device Attributes

To change the attributes of a device that the spooler subsystem uses, use the Modify Spooler Device (MSD) SCI command. The *DNOS System Command Interpreter (SCI) Reference Manual* contains a complete description of the MSD command.

12.12.3 Spooler Clean-up

An unusual series of events can cause the spooler to stop functioning without allowing you to restart it in its current state. You must reinitialize the spooler environment by performing the following steps:

1. Use the Show Output Status (SOS) command to see which files are waiting to be printed and to see the current devices available to the spooler and the class names assigned to each. Write down this information for later use.
2. Using the operator console (issue an Execute Operator Interface (XOI) command if necessary) determine the ID of the spooler job by using a List Jobs (LJ) command on all jobs.
3. Kill the spooler job by using the Kill Job (KJ) command.
4. Delete the spooler queue file by using the Delete File (DF) command and specifying the logical name S\$SDTQUE for the pathname.

- Restart the spooler job by issuing an Execute Job (XJ) command. Specify the following responses:

EXECUTE JOB

```
                SITE:
                JOB NAME:  SPOOLER
PROGRAM FILE PATHNAME:  S$UTIL
                TASK ID OR NAME:  04D
                PARM1:  1 (See previous discussion)
                PARM2:  1 (See previous discussion)
                STATION ID:  OFF
SYNONYM TABLE PATHNAME:  .S$USER.SPOOLER.SYN
                PRIORITY:  5
                JCA SIZE:  MEDIUM
```

- Use the Modify Spooler Device (MSD) command and the list made in step 1 to reassign all devices and classes to the spooler.
- Execute the Print File (PF) command for each of the files that the SOS command showed as waiting to be printed.

12.12.4 Spooler Temporary Files

The names of the spooler temporary files are based on the job ID and task ID of the user whose file is being printed. For example, the parameter xxxxxx of a spooler temporary file ID of .S\$xxxxxx contains the job or task ID. If a temporary file remains after a system crash, you can delete it to save space.

Appendix A

Keycap Cross-Reference

Generic keycap names that apply to all terminals are used for keys on keyboards throughout this manual. This appendix contains specific keyboard information to help you identify individual keys on any supported terminal. For instance, every terminal has an Attention key, but not all Attention keys look alike or have the same position on the keyboard. You can use the terminal information in this appendix to find the Attention key on any terminal.

The terminals supported are the 931 VDT, 911 VDT, 915 VDT, 940 EVT, the Business System terminal, and hard-copy terminals (including teleprinter devices). The 820 KSR has been used as a typical hard-copy terminal. The 915 VDT keyboard information is the same as that for the 911 VDT except where noted in the tables.

Appendix A contains three tables and keyboard drawings of the supported terminals.

Table A-1 lists the generic keycap names alphabetically and provides illustrations of the corresponding keycaps on each of the currently supported keyboards. When you need to press two keys to obtain a function, both keys are shown in the table. For example, on the 940 EVT the Attention key function is activated by pressing and holding down the Shift key while pressing the key labeled PREV FORM NEXT. Table A-1 shows the generic keycap name as Attention, and a corresponding illustration shows a key labeled SHIFT above a key named PREV FORM NEXT.

Function keys, such as F1, F2, and so on, are considered to be already generic and do not need further definition. However, a function key becomes generic when it does not appear on a certain keyboard but has an alternate key sequence. For that reason, the function keys are included in the table.













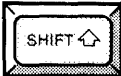
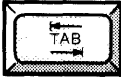





























Multiple key sequences and simultaneous keystrokes can also be described in generic keycap names that are applicable to all terminals. For example, you use a multiple key sequence and simultaneous keystrokes with the log-on function. You log on by *pressing the Attention key, then holding down the Shift key while you press the exclamation (!) key*. The same information in a table appears as *Attention!(Shift)!*.

Table A-2 shows some frequently used multiple key sequences.

Table A-3 lists the generic names for 911 keycap designations used in previous manuals. You can use this table to translate existing documentation into generic keycap documentation.

Figures A-1 through A-5 show diagrams of the 911 VDT, 915 VDT, 940 EVT, 931 VDT, and Business System terminal, respectively. Figure A-6 shows a diagram of the 820 KSR.

Table A-1. Generic Keycap Names









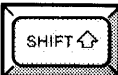












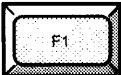





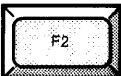





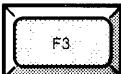





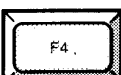





Generic Name	911 VDT	940 EVT	931 VDT	Business System Terminal	820 ¹ KSR
Alternate Mode	None				None
Attention ²		 			 
Back Tab	None	 	 	None	 
Command ²					 
Control					
Delete Character					None
Enter					 
Erase Field					 

Notes:

The 820 KSR terminal has been used as a typical hard-copy terminal with the TPD Device Service Routine (DSR). Keys on other TPD devices may be missing or have different functions.

¹On a 915 VDT the Command Key has the label F9 and the Attention Key has the label F10.

Table A-1. Generic Keypcap Names (Continued)







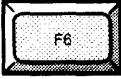








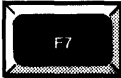
























Generic Name	911 VDT	940 EVT	931 VDT	Business System Terminal	820 ¹ KSR
Erase Input					 
Exit			 	 	
Forward Tab	 			 	 
F1					 
F2					 
F3					 
F4					 

Notes:

¹The 820 KSR terminal has been used as a typical hard-copy terminal with the TPD Device Service Routine (DSR). Keys on other TPD devices may be missing or have different functions.

2284734 (3/14)





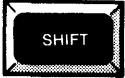















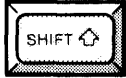




























Table A-1. Generic Keycap Names (Continued)

Generic Name	911 VDT	940 EVT	931 VDT	Business System Terminal	820 ¹ KSR
F5					 
F6					 
F7					 
F8					 
F9	 			 	 
F10	 			 	 

Notes:

¹The 820 KSR terminal has been used as a typical hard-copy terminal with the TPD Device Service Routine (DSR). Keys on other TPD devices may be missing or have different functions.

Table A-1. Generic Keycap Names (Continued)

Generic Name	911 VDT	940 EVT	931 VDT	Business System Terminal	820' KSR
F11	 			 	 
F12	 			 	 
F13	 	 	 	 	 
F14	 	 	 	 	 
Home					 
Initialize Input		 			 

Notes:

*The 820 KSR terminal has been used as a typical hard-copy terminal with the TPD Device Service Routine (DSR). Keys on other TPD devices may be missing or have different functions.














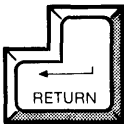




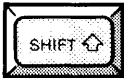



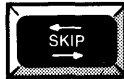







Table A-1. Generic Keypcap Names (Continued)

Generic Name	911 VDT	940 EVT	931 VDT	Business System Terminal	820 ¹ KSR
Insert Character					None
Next Character	 or 				None
Next Field	 		 	 	None
Next Line					 or
Previous Character	 or 				None
Previous Field		 			None

Notes:

¹The 820 KSR terminal has been used as a typical hard-copy terminal with the TPD Device Service Routine (DSR). Keys on other TPD devices may be missing or have different functions.

Table A-1. Generic Keycap Names (Continued)

Generic Name	911 VDT	940 EVT	931 VDT	Business System Terminal	820' KSR
Previous Line					 
Print					None
Repeat		See Note 3	See Note 3	See Note 3	None
Return					
Shift					
Skip					None
Uppercase Lock					

Notes:

¹The 820 KSR terminal has been used as a typical hard-copy terminal with the TPD Device Service Routine (DSR). Keys on other TPD devices may be missing or have different functions.

²The keyboard is typamatic, and no repeat key is needed.

228 47 34 (7/14)

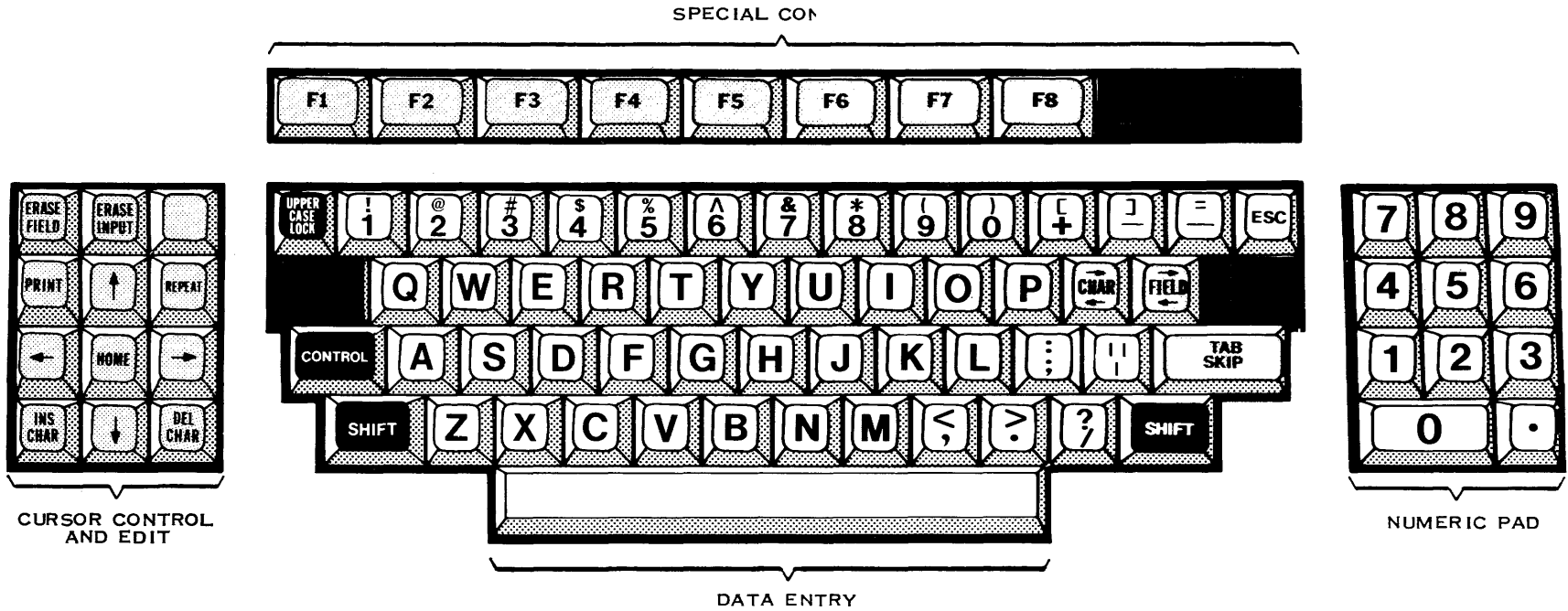
Table A-2. Frequently Used Key Sequences

Function	Key Sequence
Log-on	Attention/(Shift)!
Hard-break	Attention/(Control)x
Hold	Attention
Resume	Any key

Table A-3. 911 Keycap Name Equivalents

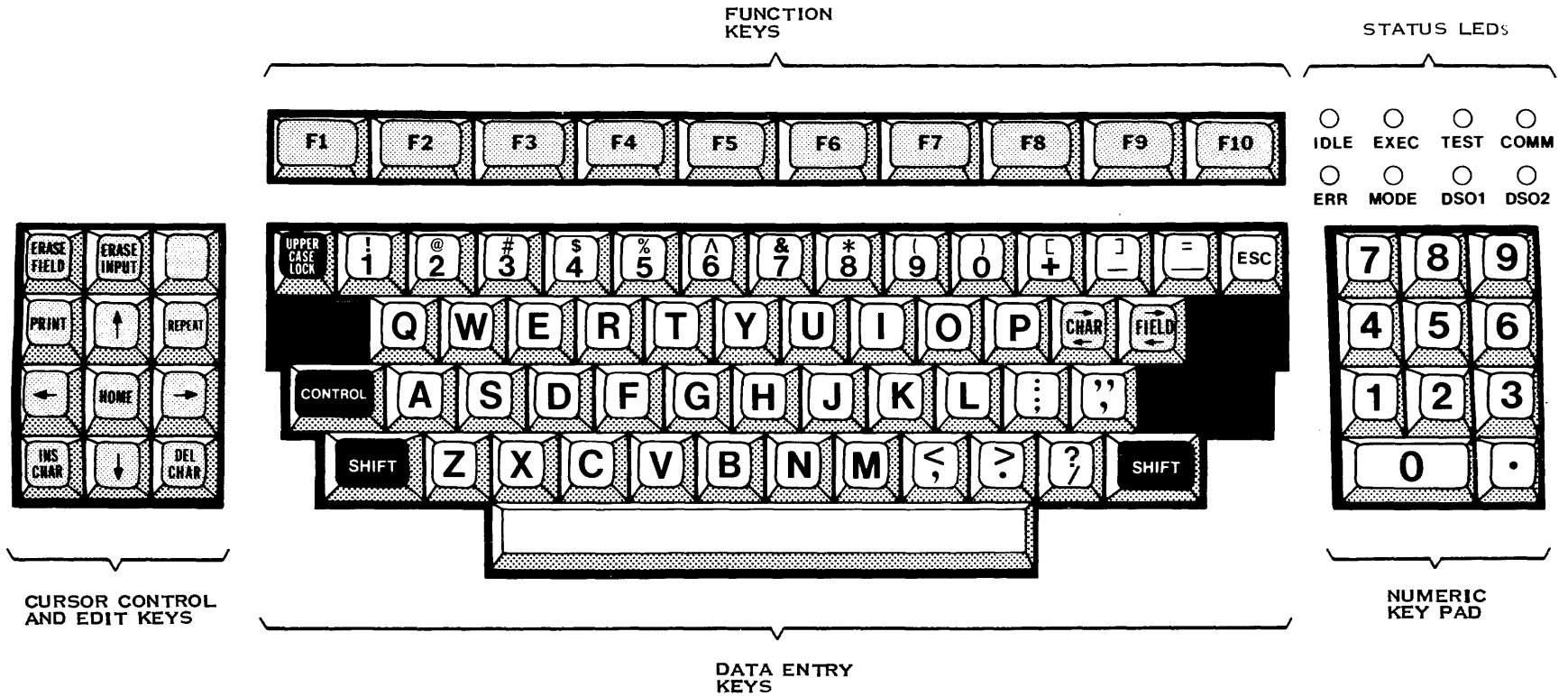
911 Phrase	Generic Name
Blank gray	Initialize Input
Blank orange	Attention
Down arrow	Next Line
Escape	Exit
Left arrow	Previous Character
Right arrow	Next Character
Up arrow	Previous Line

2284734 (8/14)



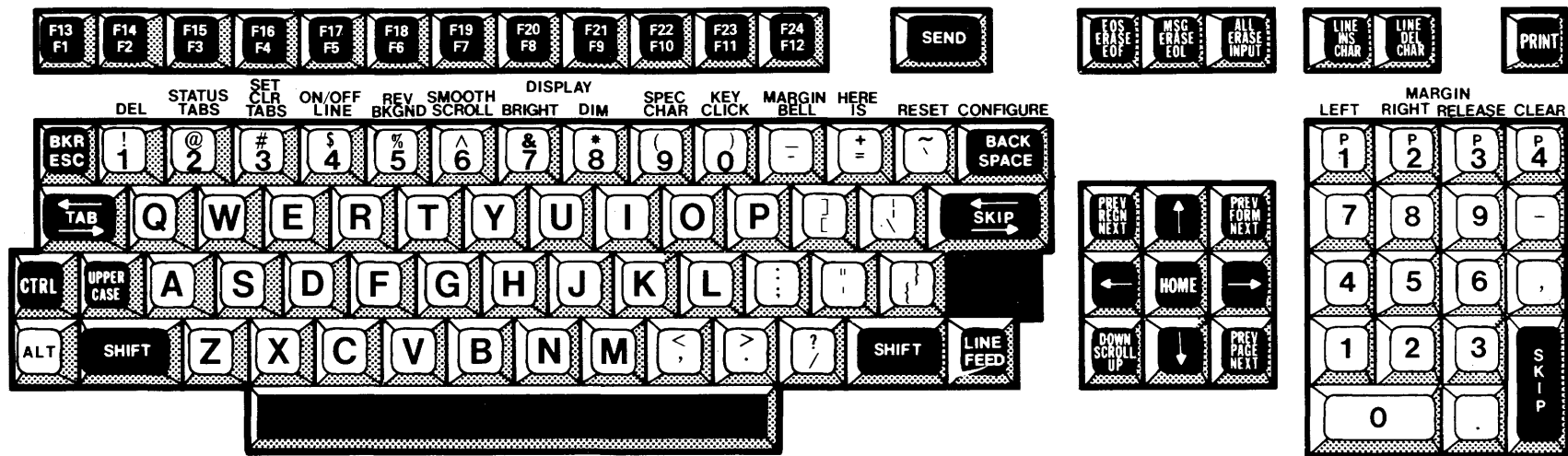
2284734 (9/14)

Figure A-1. 911 VDT Standard Keyboard Layout



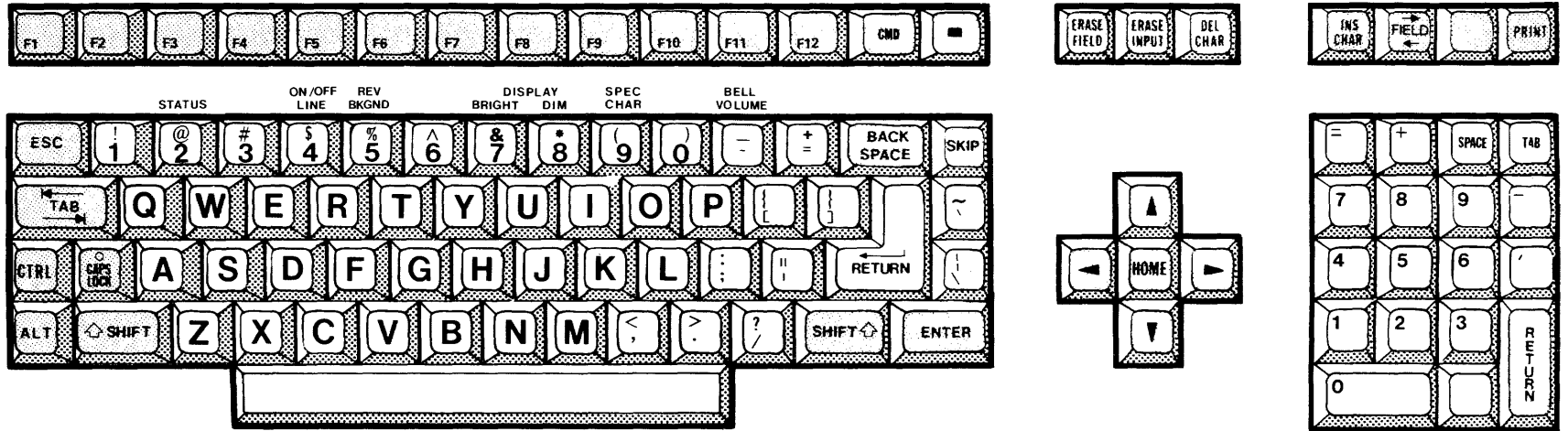
2284734 (10/14)

Figure A-2. 915 VDT Standard Keyboard Layout



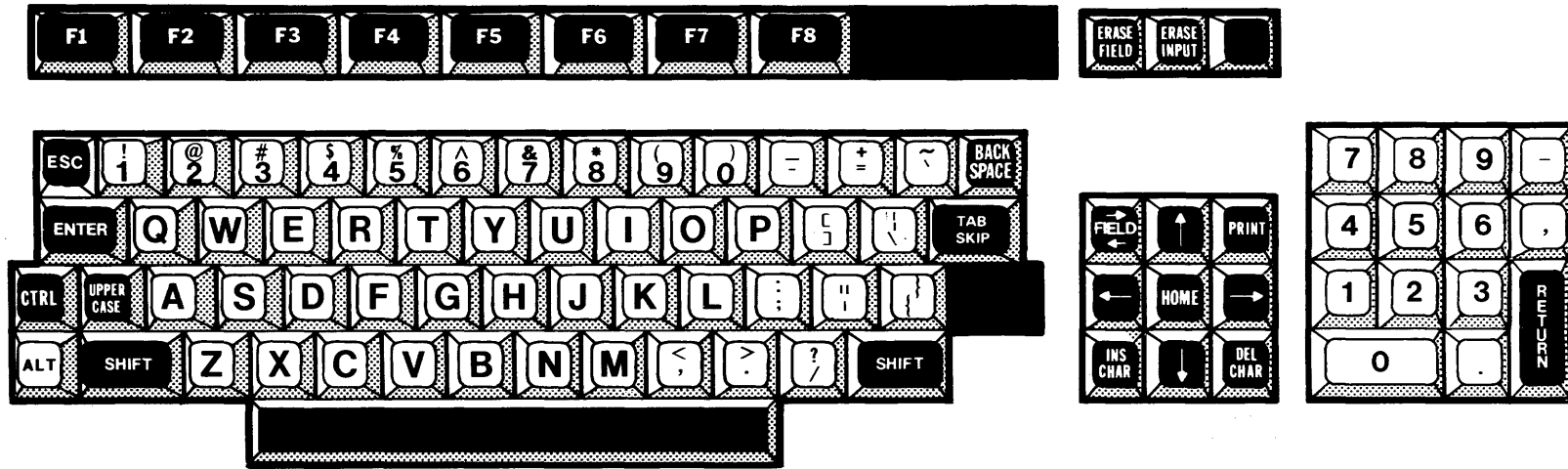
2284734 (11/14)

Figure A-3. 940 EVT Standard Keyboard Layout



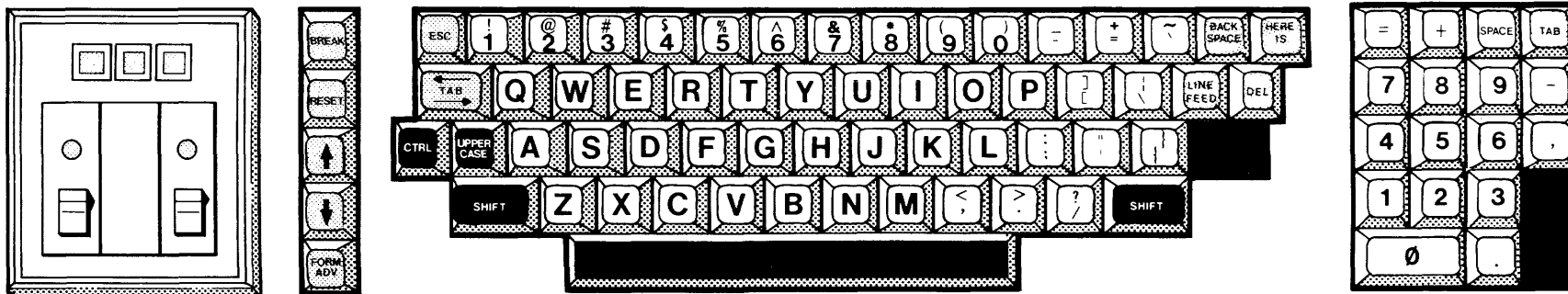
2284734 (12/14)

Figure A-4. 931 VDT Standard Keyboard Layout



2284734 (13/14)

Figure A-5. Business System Terminal Standard Keyboard Layout



2284734 (14/14)

Figure A-6. 820 KSR Standard Keyboard Layout

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.

- Abort Entry Point 5.4.2.4
- Absolute Disk Address See ADU
- Access Privileges 2.1.2.3
- Account Numbers 6.3.1
- Account Verification 6.3.1
- Accounting
 - Data 6.2
 - Files See S\$ACT1, S\$ACT2 Files
 - Subsystem Implementation 6.3
 - Subsystem Requirements 6.3.2
- ACNM Field Prompt Type 3.4.4.1
- Active Job Limit 12.2
- Add 32-Bit Integers Routine 3.10.4.1
- ADR 2.3.1.1, 2.3.1.3
- ADU 2.2.4
- Alias Descriptor Record See ADR
- Alternate Termination Routine 3.10.1.15
- Arithmetic Utility Routines 3.10.4
- Arming Character 5.4.4
- ASCHK Routine 5.5.7
- ASCK2 Routine 5.5.7
- ASCII Codes,
 - Special Language Characters T10-1
- Asynchronous:
 - Data Structure:
 - Allocation 5.6.4
 - Linkages F5-13
 - Device Support T5-2
 - DSR:
 - Building 5.8
 - Design Overview 5.6.1
 - Logic Flow F5-11
 - Structure 5.6, F5-10
 - Interrupt Processing Flow F5-12
 - Multiplexer Interrupt Decoder 5.3.3.4
- Batch:
 - Job 1.5.1, 11.4
 - Limit 12.2
 - Stream 12.10.3
- Begin Update Documentation
 - Command Procedure 12.10.1
- BEMF Utility 9.2.1
- Bind Task Routine 3.10.3, 5.5.3, 12.7.1
- Binding a Task 1.3.3
- Binding DSR Tasks 5.4.4
- Bit Map 2.2.3.2
- Blank:
 - Adjustment 2.1.3.2
 - Suppression 2.1.3.2
- Blank-Suppressed:
 - File 2.1.1.1
 - Record T2-12
- Blocked:
 - File 2.1.1.2
 - Relative Record File 2.3.3.2
- Blocking 2.1.3.1
- Blocking Buffer 2.1.3.1
- Branch Table Processor Routine 5.5.1
- BRB 5.3.2
- BRSTAT Routine 5.5.1
- BTA 1.1, 5.3.2
- BUD Command Procedure 12.10.1
- Buffer Table Area See BTA
- Buffered Request Block See BRB
- Build Expanded Message File
 - Utility 9.2., 9.2.2
- Build Message File Utility 9.2.1
- Business System Terminal
 - Keyboard Layout FA-5
- B-Tree 2.3.4.4, F2-15, F2-16
- CDE 5.4.4, 12.7.2
- CDR 2.3.1.1, 2.3.1.4
- CDT 5.4.4, 12.7.2
- Channel 1.2, 1.5.4.1
 - Activity Master/Slave 1.5.4.3
 - Activity Symmetric 1.5.4.3
 - Characteristics 1.5.4.3
 - Creation 1.5.4.2
 - Descriptor Record See CDR
 - Scope 1.5.4.1
- CIC Command 1.5.4.5
- Clock Interrupt Processor 1.3.10
- Close File Routine 3.10.2.5
- Collating Sequences 10.1
 - Supported Languages T10-3
- Command:
 - Definition Entry See CDE
 - Definition Table See CDT
 - Deletion 3.8.6
 - Library, User 3.8.4
 - Privilege Levels 3.5.1, T3-7
 - Procedure 3.2.2, 3.8
 - Design 3.8.1
 - Installation 3.8.3
 - Library, S\$CMDS 11.4
 - SCI 12.11.3
 - User-Defined 3.2
 - Processor 3.2.3, 3.8
 - Design 3.8.2
 - Installation 3.8.3
 - Interface Routines 3.9, 3.9.1
 - Snapshot Name Definitions 12.7.1
 - SND 12.7.1
 - User-Defined 3.8.4
- Common Table Area 12.5
- Compare Strings Routine 3.10.3.3
- Concatenated:
 - File 12.3
 - Sets 1.5.3.3
- Context Switch 1.1
- Controller:
 - Error 8.1.1
 - Interrupt Decoder HSR Subroutine .. 5.7.9
 - Type Codes T5-5

- Convert:
 - ASCII to Binary Integer Routine .. 3.10.3.1
 - Binary Integer to ASCII Routine .. 3.10.3.2
- Conversion Utilities .. 11.2
- Copy String Routine .. 3.10.3.4
- Country Code .. 10.2
- CPU Memory Requirements .. 11.2
- Crash Code .. 8.2.1.1
- Create:
 - IPC Channel (CIC) Command .. 1.5.4.5
 - Message Routine .. 3.10.1.13
 - Operator Message .. 3.10.6.2
- CURMAP Address .. 8.2.1.6
- Current:
 - Jobs Set .. 12.6
 - Table Use .. 12.6
- Data:
 - Block .. 2.3.4.4, F2-17
 - Buffer Address Field .. 11.3
 - Format .. 6.2.2
 - Structures .. 5.3.1
- Debugger Active Synonym .. 11.5
- DEFAULT Field Prompt Type .. 3.4.4.2
- Delayed Reentry Point .. 5.4.2.2
- Delete:
 - IPC Channel (DIC) Command .. 1.5.4.5
 - Protection .. 2.1.2.1
- Deleting Commands .. 3.8.6
- Design Criteria .. 5.4.1
- Device:
 - Information Block .. See DIB
 - Interrupt Decoder .. 5.3.3
 - I/O Operations .. 11.3
 - Service Routine .. See DSR
- DIB .. 5.3.1, 5.4.1, 5.6.4
- DIC Command .. 1.5.4.5
- Directory:
 - File .. 2.1.1.4, 2.3.1
 - Characteristics .. 2.3.1.1
 - Dump .. F2-10
 - Structure .. F2-4
 - Overhead Record Format .. F2-5
 - Structure .. F2-3
- Disk:
 - Access Frequency .. 2.2.2
 - File Structure .. 2.3
 - Format Information .. T2-1
 - Organization .. 2.2
 - Physical Organization .. 2.2.3
 - Space .. 2.1.3.2
 - Allocation .. 2.2.2
 - Volume Information .. 2.2.3.1, F2-1
- Disk Fragmentation, Eliminating .. 12.7.5
- Divide 32-Bit Integers Routine .. 3.10.4.4
- DNOS:
 - CPU Memory Requirements .. 11.2
 - Environment .. 11.2
 - Job Architecture .. 12.2
 - Physical Layout .. F1-1
 - SVCs .. 11.6
 - Subsystems .. 1.5
- System:
 - File Names .. T11-1
 - Generation .. 11.4
- DSR .. 5.1, 5.4, 5.4.3
 - See also Asynchronous DSR
 - Debugging Techniques .. 5.9
 - Design Overview, Asynchronous .. 5.6.1
 - Entry Points .. 5.4.2
 - Installation .. 5.8
 - Link Control Stream .. 5.8
 - Listing Example .. 5.10, F5-14
 - Memory Map .. F5-8
 - Structure .. F5-7
 - Support Routines .. 5.5
 - Task Activation .. 5.4.4
 - Templates .. 5.3.1
- DSR/TSR Entry Points .. T5-1
- DX10 System File Names .. T11-1
- DX10-to-DNOS .BID/.OVLY
 - Converter .. 12.11.3
- Dynamic:
 - Display of Tasks .. 12.6
 - Modification of Run Time
 - Parameters .. 1.3.8
 - Priority Tasks .. 1.3.7
- ELEMENT Field Prompt Type .. 3.4.4.3
- Enable/Disable Status Change Notification
 - HSR Subroutine .. 5.7.4
- End:
 - Operator Interface Subsystem Interface
 - Session
 - Routine .. 3.10.6.4
 - Update Documentation Command
 - Procedure .. 12.10.2
- End-of-File .. See EOF
- ENDRCD Routine .. 5.5.2
- Entry Point:
 - Abort .. 5.4.2.4
 - Hardware Interrupt .. 5.4.2.1
 - Initial Request .. 5.4.2.7
 - Power-Up Initialization .. 5.4.2.3
 - Priority Scheduler .. 5.4.2.6
 - Time-Out .. 5.4.2.5
- Entry Points:
 - DSR .. 5.4.2
 - DSR/TSR .. T5-1
- EOF .. 2.1.3.6
- EOR Processor Routine .. 5.5.2
- Error:
 - Codes, Internal .. 9.2.1
 - Interrupt .. 5.4.2.1
 - Message Interface .. 9.4
 - Reporting .. 9.1
- ERSTAT Routine .. 5.5.2
- EUD Command Procedure .. 12.10.2

- Exclusive:
 - All Privilege 2.1.2.3
 - Write Privilege 2.1.2.3
- Execute:
 - Batch Job (XBJ) Command 1.5.1.2
 - Job (XJ) Command 1.5.1.2
- Executing:
 - Task 8.2.1.2
 - Task JSB 8.2.1.3
 - Workspace 8.2.1.8
- Execution Priorities 1.3.7
- Expanded:
 - Message Files 9.2.1
 - Message File Utility 9.3.2
- Expanding Files 2.1.3.5
- Expansion Chassis:
 - Interrupt 5.3.3.3
 - Interrupt Decoder F5-3
- Expert Mode 3.8.5
- Extended:
 - Call Block Flag 11.3
 - Operations See XOPs
- Extending SCI 3.2
- Failure Location 8.2.1.4
- Fast VDT Mode 12.4
- FDB 1.4.3
- FDR 2.3.1.1, 2.3.1.2
- Field Prompt 3.4.4
 - List 3.5.1
 - Types T3-4
- File:
 - Characteristics 2.1.3
 - Concatenated 12.3
 - Concatenation Restrictions 1.5.3.3
 - Descriptor Block See FDB
 - Descriptor Record See FDR
 - Directory 2.1.1.4, 2.3.1
 - Characteristics 2.3.1.1
 - Indicator 9.2.1
 - I/O Operations 11.3
 - Name 2.3.1.1
 - Organization 2.1
 - Protection 2.1.2
 - Sharing 2.1.2
 - Types 2.1.1
- Flag Word 8.2.2.1, F8-4
- Flash Crash Routine 8.1.2
- General Information Block F8-2
- Generic Keycap Names .. Appendix A, TA-1
- Get:
 - Buffer Routine 5.5.9
 - Event Character Routine 5.5.6
 - Parameter Routine 3.10.1.3
 - Queued Character Routine 5.5.5
 - Synonym Value Routine 3.10.1.7
 - TCA Routine 3.10.1.1
 - Terminal Status Routine 3.10.1.5
 - 20-Bit TILINE Address Routine 5.5.11
- GETC Routine 5.5.5
- GTADDR Routine 5.5.11
- Halt Job (HJ) Command 1.5.1.2
- Hard Break Key Sequence 11.2
- Hardware:
 - Interrupt:
 - Entry Point 5.4.2.1
 - Processing F5-9
 - Map Option 1.1.1
 - Service Routine See HSR
 - Trap 8.2.1.9
- Hash Placement KIF 11.2
- HJ Command 1.5.1.2
- HSR 5.6, 5.6.1, 5.6.4
 - Baud Rate Codes T5-4
 - Common Subroutines 5.7
 - Object Modules T5-3
 - Required Information 5.7
 - Subroutine Classes 5.7
- Idle State 1.3.2
- Illegal Interrupt Routine 5.3.3.6, F5-6
- Image File 2.1.1.4
- Immediate Write Option 2.1.3.3
- Indicator Lights 8.1.1
- Information Interchange Codes 10.3
- Initial:
 - Program Load See IPL
 - Request Entry Point 5.4.2.7
- Initialization Problems, System 8.1
- Initialize:
 - Mailbox Interface Routine 3.10.7.1
 - Operator Interface Routine 3.10.6.1
 - System Data Base Routine 3.10.1.2
- Input Interrupt 5.4.2.1
- INT Field Prompt Type 3.4.4.4
- Interface:
 - Routine Buffers 3.9.2
 - Routines 3.10
 - SCI 3.10.1
- Internal:
 - Error Codes 9.2.1
 - Interrupt Processor 1.3.11
- International:
 - Data 10.2
 - Languages 10.1
- Internationalizing Messages 10.5
- Interprocess Communication See IPC
- Interrupt:
 - Decoder:
 - Asynchronous Multiplexer 5.3.3.4
 - Device 5.3.3
 - Expansion Chassis F5-3
 - Multiple Devices F5-2
 - Return Routine F5-5
 - Routines 5.3.3
 - Single Device F5-1
 - Expansion Chassis 5.3.3.3
 - Mask 5.4.2

- Multiple-Device 5.3.3.2
- Processing F5-9
- Service Routine See ISR
- Single-Device 5.3.3.1
- Intertask Message Queue 11.6, 12.5
- IOFCDT Routine 5.5.3
- IOGEC Routine 5.5.6
- IOGUB Routine 5.5.9
- IOMPIN Routine 5.5.10
- IOMPOT Routine 5.5.8
- IPC 1.5.4, 12.5
 - SCI Commands 1.5.4.5
 - Supervisor Calls 1.5.4.4
- IPL 1.3.1, 1.4, 8.2
- IRB 5.3.2
- ISR 5.4.2.1, 5.6, 5.6.1, 5.6.2
 - Functions 5.4.2.1
- I/O:
 - Request Block See IRB
 - Subsystem 5.3
- Japanese Character Codes T10-2
- JCA 1.1.2.1
- JISCIJ T10-2
- Job:
 - Architecture 11.2, 12.2
 - Control Capabilities 1.5.1.2
 - Global Data 8.2.2.2
 - Initialization 6.2.1
 - Management Request SVC 1.5.1.1
 - Status Block (JSB) 1.1.2.1
 - Structure 1.1.2.1
 - Temporary Files 1.5.3.2, 11.2
 - Termination 6.2.1
- Jobs 1.5.1
- JSB 1.1.2.1, 8.2.2
 - Executing Task 8.2.1.3
 - List 8.2.2.2, F8-3
- KDR 2.3.1.1, 2.3.1.5
- Kernel 1.1
- Key:
 - Descriptor Record See KDR
 - Sequences TA-2
- Keyboard:
 - Bid 12.2, 12.7.2
 - Layout:
 - Business System Terminal FA-5
 - 820 KSR FA-6
 - 911 VDT FA-1
 - 915 VDT FA-2
 - 931 VDT FA-4
 - 940 EVT FA-3
 - Sequence, Predefined 12.7.2
 - Status Block See KSB
- Keycap Name Equivalents, 911 VDT ... TA-3
- Keycap Names, Generic .. Appendix A, TA-1
- Keyword List 3.3
- KIF 2.1.1.3, 2.3.4
 - Collating Sequence 10.4
 - Disk Usage 2.3.4.6
 - Hash Placement 11.2
 - Keys 2.3.4.1
 - Logical Record 2.3.4.5
 - Manager 10.4
 - Multifile 12.3
 - Sets 1.5.3.3
 - Records 2.3.4.2
 - Sequential Placement 11.3
 - Structure 2.3.4.4, F2-13
- Kill Job (KJ) Command 1.5.1.2
- KJ Command 1.5.1.2
- KSB 5.4.1
- Language:
 - Line 3.3
 - Processors 10.5
- Library, User Command 3.8.4
- Limit Registers 1.1.1
- Link Editor Control Stream 3.8.3
- Load, TILINE 8.1.1
- Loader:
 - Flashing Crash Codes, System T8-2
 - Phases, System T8-1
- Local Display File Routines 3.10.2
- Locking, Record 2.1.2.2
- Logical:
 - Address Space 1.1.1
 - Name 3.1, 3.4.2, 12.3
 - Names 1.5.3.1
 - Name Table 3.1, 11.3, 12.4
 - Records 2.1.3.1
 - Record Size 2.1.3.1
- LP\$1 Function 11.3
- LUNO:
 - Types 12.7.3
 - 00 11.2, 11.3
- Mailbox Subsystem
 - Interface Routines 3.10.7
- Map 12.6
 - File 1.1.1
 - In Buffer Routine 5.5.10
 - Option 1.1.1
 - Out Current Buffer Routine 5.5.8
- Master/Slave:
 - Channel 1.5.4.3
 - Owner Task 1.5.4.3
- MB\$INT Routine 3.10.7.1
- MB\$RCV Routine 3.10.7.3
- MB\$RLS Routine 3.10.7.4
- MB\$SND Routine 3.10.7.2
- Memory:
 - Mapping 1.1.1
 - Resident User Tasks 1.1
 - Statistics 12.6

- Message:
 - File Utilities 9.3.1, F9-1
 - Processing Synonyms T3-3
- Migrating Between DX10 and DNOS ... 11.1
- Modify Job Priority Command 1.5.1.2
- MJP Command 1.5.1.2
- Monitoring Utilities 12.6
- Multifile KIF 12.3
- Multiple-Device Interrupt 5.3.3.2
- Multiple Devices, Interrupt Decoder ... F5-2
- Multiplexing Hidden Request Queue .. 5.4.5
- Multiply 32-Bit Integers Routine 3.10.4.3
- Multivolume File Sets 1.5.3.3, 1.5.3.4

- Name:
 - Definition 3.4.3
 - Stages 1.5.3.5
 - Management 1.5.3
 - Name Manager, SVC 3.4.1
 - Manager Task 3.1
- NAME Field Prompt Type 3.4.4.5
- Names, Reserved 9.2.1
- Nonreplicable Tasks 11.2
- Nucleus 1.1

- OI\$BGN Routine 3.10.6.1
- OI\$COM Routine 3.10.6.2
- OI\$END Routine 3.10.6.3
- OI\$WAT Routine 3.10.6.2
- Open:
 - File Routine 3.10.2.1
 - File Specifying User ID Routine .. 3.10.2.2
 - Operation 11.3
- Operator Interface Routines 3.10.6
- Output a Character HSR Subroutine ... 5.7.5
- Output Interrupt 5.4.2.1

- Partial Bit Map 1.1, 2.2.3.2, T2-2
- PDT 5.3.1, 5.4.1
- Special Extension 5.8
- Performance:
 - Monitoring 12.6
 - Optimization 12.7
- PF Command 11.4
- Physical:
 - Device Table See PDT
 - Record Length 2.1.3.1
 - Records 2.1.3.1
- Power-Up Initialization:
 - HSR Subroutine 5.7.1
 - Entry Point 5.2.3
- Primary KIF Key 5.4.2.3
- Primitives 3.5
- Print File Command 11.4
- Priority Scheduler Entry Point 5.4.2.6
- Procedure Library 3.2.2
- Program:
 - File 2.1.1.4
 - Image Loader 1.4.2
 - Errors 8.1.2
- Put TCA Routine 3.10.1.11
- PUTCBF Routine 5.5.4
- PUTEBF Routine 5.5.4

- Queue 1.1.2.3
 - Character Routine 5.5.4
 - Event Character Routine 5.5.4
 - Servers 1.1.2.3
 - Structure F1-4

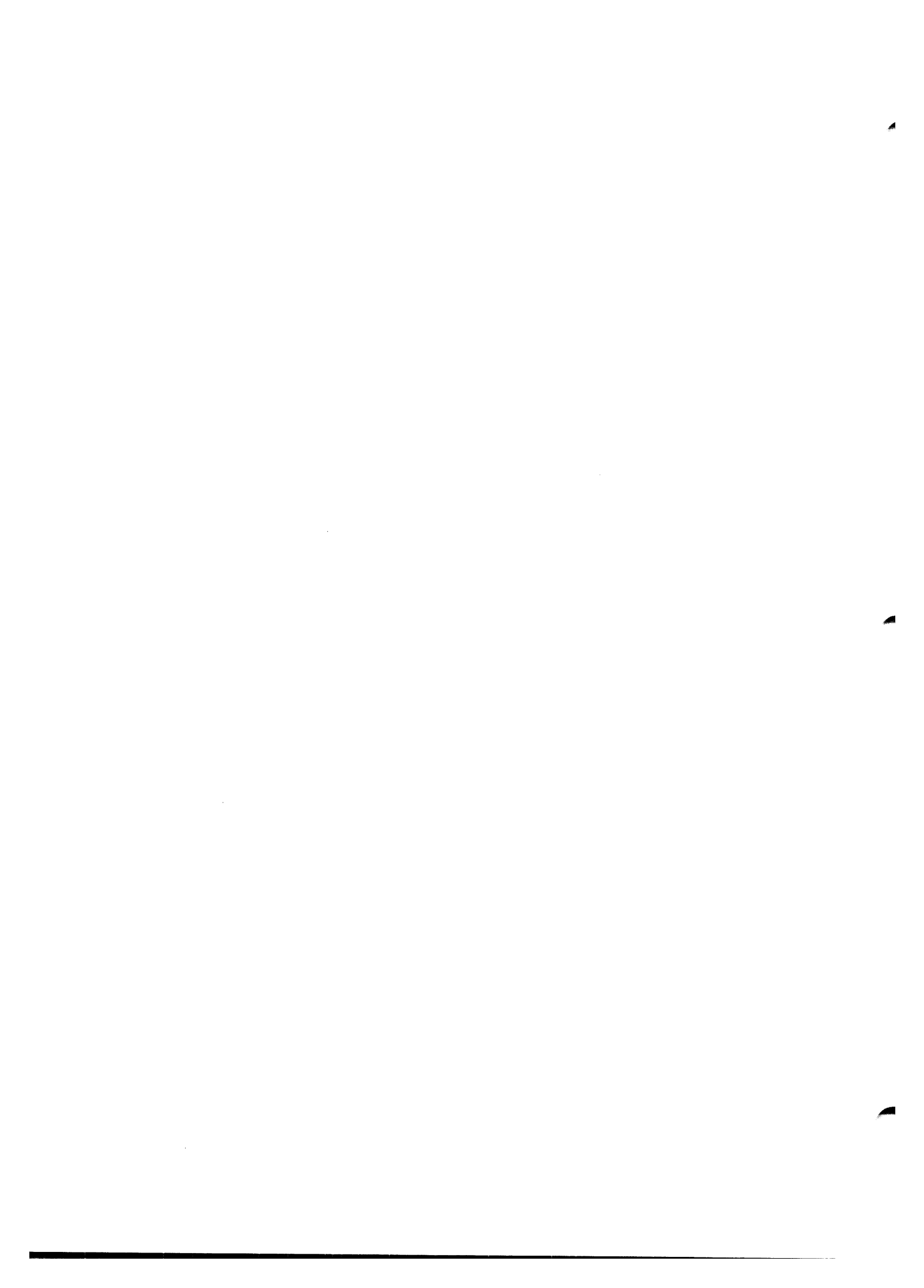
- RANGE Field Prompt Type 3.4.4.6
- RDB T4-1
- Read:
 - HSR Subroutine 5.7.3
 - Only Privilege 2.1.2.3
 - Input Signal or Function Operational Parameters and Information HSR Subroutine 5.7.7
- Receive Mail Routine 3.10.7.3
- Record:
 - Blocking 2.1.3.1
 - Locking 2.1.2.2
 - Logical 2.1.3.1
 - Physical 2.1.3.1
- Reentry Point, Delayed 5.4.2.2
- Relative Record:
 - File 2.1.1.2, 2.3.3, 12.3.7
 - File, Blocked 2.3.3.2
 - File, Unblocked 2.3.3.1
- Release:
 - Mailbox Routine 3.10.7.4
 - TCA Routine 3.10.1.11
- Report TILINE Error Routine 5.5.13
- Request:
 - Descriptor Block See RDB
 - Flow 5.3.2
 - Information Block See RIB
 - Operation Code 5.3.2
 - Time Interval Notification HSR Subroutine 5.7.8
- Reserved Names 9.2.1
- Resume Job (RJ) Command 1.5.1.2
- Return Routine 5.3.3.5
- Return Time and Date Routine 3.10.1.10
- RE-ENTER-ME Routine 5.4.2.2
- RJ Command 1.5.1.2
- ROM Loader 1.4.1
- Errors 8.1.1
- Run-Time Routines 11.5

- SAT Structures 12.7.5
- SCI 3.1
 - Access to Logical Names and Synonyms F3-2
 - Batch Mode 3.1
 - Command 3.3
 - Procedures 11.5, 12.11.3
 - Features 12.4
 - Interactive Mode 3.1
 - Interface Routines 3.10.1, 11.5

- Language:
 - Syntax 3.3
 - Maintained Synonyms T3-2
 - Modes of Operation F3-1
 - Primitive 3.2.1, 3.3, 11.5
 - Batch Stream 3.6
 - Error Processing 3.7
 - Primitives 3.5, T3-5, T3-6
 - Run-Time Routines 11.5
 - Sessions, Structuring 12.7.1
 - Special Characters T3-1
 - Task 11.5
 - User Interface 11.4
 - Variables 3.4
- SCS Command 1.5.4.5
- Search Name Correspondence
 - Table Routine 3.10.1.8
- Secondary Allocation
 - Table Structures 12.7.5
- Security Option Initialization 1.4
- Segment Management 1.5.2
- Send Mail Routine 3.10.7.2
- Sequential:
 - File 2.3.2
 - Files 2.1.1.1
 - Placement KIF 11.3
 - Record Placement 2.3.4.4, F2-14
- Set:
 - Channel Speed HSR Subroutine ... 5.7.6.1
 - Data Character Format
 - HSR Subroutine 5.7.6.2
 - Device Parameters Suboperation ... 5.4.4
 - Synonym Value Routine 3.10.1.6
- Shared Privilege 2.1.2.3
- Show:
 - Channel Status (SCS) Command .. 1.5.4.5
 - Job Status (SJS) Command 1.5.1.2
- Single-Device Interrupt 5.3.3.1
- Single Device, Interrupt Decoder F5-1
- SJS Command 1.5.1.2
- Snapshot Name Definitions
 - Command 12.7.1
- SND Command 12.7.1
- Special:
 - Language Characters 10.1, T10-1
 - Usage File Protection 2.1.2.4
 - Usage Files 2.1.1.4
 - Workspaces 8.2.1.10
- Split List Into Components
 - Routine 3.10.1.9
- Spooler 12.3, 12.12
 - Banner Sheet File 12.12.1.1
 - Clean-Up 12.12.3
 - Device Attributes 12.12.2
 - Directory 12.12.1
 - Interface Routine 3.10.5
 - Job 12.12
 - Queue File 12.12.1.2
 - Temporary Files 12.12.4
- Static Priority Tasks 1.3.7
- Station Local LUNOs 11.3
- Status:
 - Interrupt 5.4.2.1
 - Register 8.2.1.5
- STRING Field Prompt Type 3.4.4.7
- String Utility Routines 3.10.3
- Subroutines See HSR Subroutines, Routines
- Subtract 32-Bit Integers Routine ... 3.10.4.2
- SVC Processor 4.1, 4.2, 4.3
- SVC Call Block 4.2.1
- SVC Definition Tables 4.2.2
- SVC Sysgen Requirements 4.2.4
- SVCs, Differences Under DNOS 11.6
- Symmetric Channel 1.5.4.3
- Synonym 3.1
 - Evaluation 3.4.1
 - Name Table 12.4
 - Table 3.1, 11.4
- Synonyms 3.4.1
- System:
 - Command Interpreter See SCI
 - Command Procedures 12.10
 - Components 1.1
 - Configuration Utility 12.8
 - Crash 8.1.1
 - Dumps 8.2.3
 - Forcing 8.2.4
 - Problems 8.2
 - Routine 1.3.12
 - Disk 1.1.2.4
 - File Names 11.2, T11-1
 - Files 1.1.2.4
 - Flow 1.3
 - Initialization Problems 8.1
 - Interrupt Decoder Routines 5.3.3
 - Loader 1.4.3
 - Error F8-1
 - Flashing Crash Codes T8-2
 - Phases T8-1
 - Loaders 1.4
 - Log Device 11.2
 - Memory Mapping 1.1.1
 - Patch Area 8.2.1.7
 - Restart Task 1.4
 - Structure 1.1
 - Tables 1.1
 - Table Sizes 12.9
 - Tasks 1.1
- S\$:
 - Files 1.1.2.4
 - Interface Routines 3.1
 - Routines 3.9, 11.5
- \$\$BIDT Routine 3.10.1.3, 5.5.3, 12.7.1
- \$\$CLOS Routine 3.10.2.5
- \$\$CMMSG Routine 3.10.1.13
- \$\$CRASH File 11.2
- \$\$GTCA Routine 3.10.1.1

\$\$IADD Routine	3.10.4.1	Transfer:	
\$\$IASC Routine	3.10.3.2	PDT Information to Task Memory	
\$\$IDIV Routine	3.10.4.4	Routine	5.5.12
\$\$IMUL Routine	3.10.4.3	Vectors	1.1
\$\$INT Routine	3.10.3.1	Trap, Hardware	8.2.1.9
\$\$ISUB Routine	3.10.4.2	TSB	1.1.2.2
\$\$MAPS Routine	3.10.1.7	TSR	5.6, 5.6.1, 5.6.2
\$\$NEW Routine	3.10.1.2		
\$\$OPEN Routine	3.10.2.1	Unblocked:	
Error	11.5	File	2.1.1.2
\$\$OPNS Routine	3.10.2.2	Relative Record File	2.3.3.1
\$\$PARM Routine	3.10.1.4	Undetected Write Error, Preventing	2.1.3.3
\$\$PTCA Routine	3.10.1.11	Unit Select Error	8.1.1
\$\$RTCA Routine	3.10.1.12	Unsolicited Interrupt	5.4.2.1
\$\$SCOM Routine	3.10.3.3	User Command Library	3.8.4
\$\$SCPY Routine	3.10.3.4	User Flags Byte	11.3
\$\$SETS Routine	3.10.1.6	User IDs	11.2, 11.5, 12.11
\$\$SNCT Routine	3.10.1.8	User-Defined:	
\$\$SPLR Routine	3.10.5	Commands	3.8.4
\$\$SPLT Routine	3.10.1.9	DSRs	11.7
\$\$STAT Routine	3.10.1.5	SVC Processors	11.7
\$\$STOP Routine	3.10.1.15	System Software	11.7
\$\$TAD Routine	3.10.1.10		
\$\$TERM Routine	3.10.1.14	Wait for Operator Response	
\$\$WEOL Routine	3.10.2.4	Routine	3.10.6.3
\$\$WRIT Routine	3.10.2.3	Write:	
Table Overflow, Preventing	3.8.1	EOL to File Routine	3.10.2.4
Task:		Operational Parameters HSR	
Activation	1.3.3	Subroutine	5.7.6
Bidding a	1.3.3	Output Signal or Function HSR	
Characteristics	8.2.2.1	Subroutine	5.7.2
Flag	8.2.2.1	Protection	2.1.2.1
JSB, Executing	8.2.1.3	To File Routine	3.10.2.3
Priority	1.3.7		
Run IDs	11.6	XANAL	8.2
Scheduler	1.3.4	Commands	T8-3
States	8.2.2, 8.2.2.1, F8-3	Listing	8.2.1
Status Block (TSB)	1.1.2.2	XBJ Command	1.5.1.2
Structure	1.1.2.2, F1-3	XFERM Routine	5.5.12
Termination	1.3.9, 6.2.1	XJ Command	1.5.1.2
Temporary Files	2.1.3.4	XOP:	
Terminal:		Processing	1.3.6
Local File	See TLF	Vectors	8.2.1.9
Service Routine	See TSR	XOPs	1.1
Terminate and Return to SCI		XPROCCVT Converter	12.11.3
Routine	3.10.1.13		
Termination, Task	1.3.9	YESNO Field Prompt Type	3.4.4.8
Text Editor Active Synonym	11.5		
TILERR Routine	5.5.13	\$\$CC Synonym	11.5
TILINE:		\$\$DA Synonym	11.5
Load	8.1.1	\$\$EA Synonym	11.5
Logical Address Space	1.1.1		
Peripheral Control Space	See TPCS	.BID Primitive	3.5.9
Time Slicing	1.3.5	.DATA Primitive	3.5.13
Time-Out Entry Point	5.4.2.5	.DBID Primitive	3.5.10
Timing Interrupt	5.4.2.1	.ELSE Primitive	3.5.2
TLF	3.1	.ENDIF Primitive	3.5.2
TPCS	1.1.1	.EOD Primitive	3.5.13
Status	8.1.1	.EOP Primitive	3.5.1
		.EVAL Primitive	3.5.6

.EXIT Primitive	3.5.5
.IF Primitive	3.5.2
.LOOP Primitive	3.5.7
.MENU Primitive	3.5.17
.MESSAGES.TEXT Directory	9.2.1, 10.5
.OPTION Primitive	3.5.16
Use With International ASCII	10.4
.PROC Primitive	3.5.1
.PROMPT Primitive	3.5.3
.QBID Primitive	3.5.11
.RBID Primitive	3.5.12
.REPEAT Primitive	3.5.7
.SCI990 Library	3.9.1
.SHOW Primitive	3.5.18
.SPLIT Primitive	3.5.8
.STOP Primitive	3.5.14
.SVC Primitive	3.5.19
Disallowed SVCs with	T3-8
.SYN Primitive	3.5.4
.\$ACCCHN Channel	1.2
.\$ACT1, .\$ACT2 Files	1.1.2.4, 6.1
.\$CDT Directory	1.1.2.4
.\$CDTQUE Directory	1.1.2.4
.\$CLF File	1.1.2.4
.\$CMDS:	
Command Procedure Library	11.4
Directory	1.1.2.4, 12.9
.\$CRASH File	1.1.2.4
.\$DIAG File	1.1.2.4
.\$DSTCHN Channel	1.2
.\$EXPMSG Directory	1.1.2.4
.\$IPL File	1.1.2.4, 1.4.3
.\$ISBTCH File	1.1.2.4, 1.4
.\$ISLIST File	1.1.2.4
.\$LANG File	1.1.2.4
.\$LOG1 File	1.1.2.4
.\$LOG1,.\$LOG2 Files	1.1.2.4
.\$MAIL Channel	1.2
.\$MSG Directory	1.1.2.4, 9.2
.\$MVI File	1.1.2.4
.\$OPER Channel	1.2
.\$PWCS File	1.1.2.4
.\$ROLLA, .\$ROLLD Files	1.1.2.4
.\$SCA File	1.1.2.4
.\$SDTQUE Directory	12.12.1
.\$SDTQUE,.\$BANNER File	12.12.1.1
.\$SECURE File	1.1.2.4
.\$SGU\$ Directory	1.1.2.4
.\$SHARE File	1.1.2.4
.\$SHARED File	1.1.2.4
.\$SPOOL Channel	1.2
.\$SYSLIB Directory	1.1.2.4
.\$SYSTEM Directory	1.1.2.4, 12.9
.\$SYSTEM,.\$HSTRY File	12.9
.\$USER Directory	1.1.2.4
.\$UTIL File	1.1.2.4
.\$XBJ Channel	1.2
.\$UTIL	6.3.3
.UNTIL Primitive	3.5.7
.USE Primitive	3.5.15
.WHILE Primitive	3.5.7
7-Bit Character Check Routine	5.5.7
8-Bit Character Check Routine	5.5.7
820 KSR Keyboard Layout	FA-6
911 VDT:	
Keyboard Layout	FA-1
Keycap Name Equivalents	TA-3
915 VDT Keyboard Layout	FA-2
931 VDT Keyboard Layout	FA-4
940 EVT Keyboard Layout	FA-3



FOLD



NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

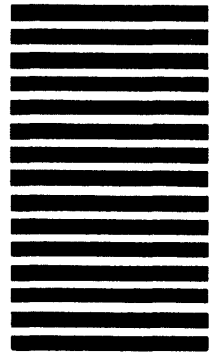
BUSINESS REPLY MAIL

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