DNOS

System Design Document

Part No. 2270512-9701 *B
15 November 1983

TEXAS INSTRUMENTS

MANUAL REVISION HISTORY

DNOS System Design Document (2270512-9701)

Original Issue	1 August 1981
Revision	1 October 1982
Revision	15 November 1983

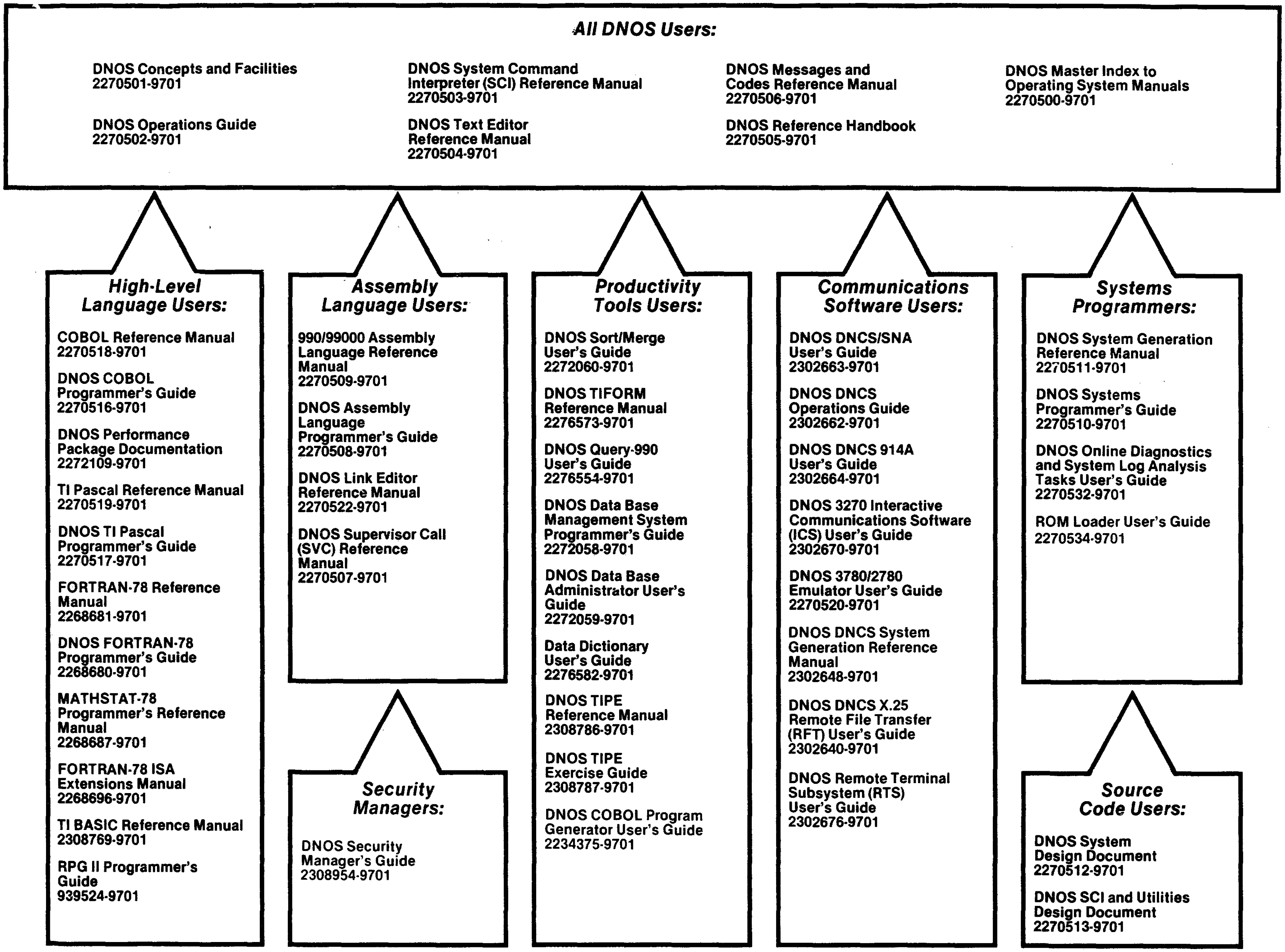
The total number of pages in this publication is 706.

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

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

DNOS Software Manuals

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



All DNOS Users:

DNOS Concepts and Facilities
2270501-9701

DNOS Operations Guide
2270502-9701

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

DNOS Text Editor Reference Manual
2270504-9701

DNOS Messages and Codes Reference Manual
2270506-9701

DNOS Reference Handbook
2270505-9701

DNOS Master Index to Operating System Manuals
2270500-9701

High-Level Language Users:

COBOL Reference Manual
2270518-9701

DNOS COBOL Programmer's Guide
2270516-9701

DNOS Performance Package Documentation
2272109-9701

TI Pascal Reference Manual
2270519-9701

DNOS TI Pascal Programmer's Guide
2270517-9701

FORTRAN-78 Reference Manual
2268681-9701

DNOS FORTRAN-78 Programmer's Guide
2268680-9701

MATHSTAT-78 Programmer's Reference Manual
2268687-9701

FORTRAN-78 ISA Extensions Manual
2268696-9701

TI BASIC Reference Manual
2308769-9701

RPG II Programmer's Guide
939524-9701

Assembly Language Users:

990/99000 Assembly Language Reference Manual
2270509-9701

DNOS Assembly Language Programmer's Guide
2270508-9701

DNOS Link Editor Reference Manual
2270522-9701

DNOS Supervisor Call (SVC) Reference Manual
2270507-9701

Security Managers:

DNOS Security Manager's Guide
2308954-9701

Productivity Tools Users:

DNOS Sort/Merge User's Guide
2272060-9701

DNOS TIFORM Reference Manual
2276573-9701

DNOS Query-990 User's Guide
2276554-9701

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

DNOS Data Base Administrator User's Guide
2272059-9701

Data Dictionary User's Guide
2276582-9701

DNOS TIPE Reference Manual
2308786-9701

DNOS TIPE Exercise Guide
2308787-9701

DNOS COBOL Program Generator User's Guide
2234375-9701

Communications Software Users:

DNOS DNCS/SNA User's Guide
2302663-9701

DNOS DNCS Operations Guide
2302662-9701

DNOS DNCS 914A User's Guide
2302664-9701

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

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

DNOS DNCS System Generation Reference Manual
2302648-9701

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

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

Systems Programmers:

DNOS System Generation Reference Manual
2270511-9701

DNOS Systems Programmer's Guide
2270510-9701

DNOS Online Diagnostics and System Log Analysis Tasks User's Guide
2270532-9701

ROM Loader User's Guide
2270534-9701

Source Code Users:

DNOS System Design Document
2270512-9701

DNOS SCI and Utilities Design Document
2270513-9701

DNOS Software Manuals Summary

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Online Diagnostics and System Log Analysis Tasks User's Guide
Explains how to execute the online diagnostic tasks and the system log analysis task and how to interpret the results.

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 DNOS system design document contains the information that is needed to understand the operation of the system but is not provided in other DNOS manuals. The document describes the flow of control of the operating system in general and of each of its subsystems in particular. It also includes data structure pictures, link streams, and directory information for DNOS modules. Revisions made to this manual since DNOS 1.1 are marked with revision bars in the margins.

This manual is divided into the following sections:

Section

1 How to Use the Design Document -- Explains how to use this manual.

2 Overview of DNOS -- Discusses the general features of the DNOS system.

3 Naming and Coding Conventions -- Explains the conventions used in writing DNOS modules.

4 DNOS Structure and Nucleus Functions -- Discusses the overall structure of DNOS, common interface routines, map file usage, major data structures, and queue server structures.

5 IPL and System Loaders -- Describes the process of Initial Program Load and the structures that support system loading.

6 SVC Processing -- Discusses the preprocessing done by the system, the several paths of control through supervisor call (SVC) processing, and how new SVC processors can be added to DNOS.

7 Segment Management -- Explains the structures, function, and use of segment management facilities of DNOS.

8 Job Management -- Discusses the job management flow of control and the data structures that support job management.

9 Program Management -- Describes the flow of control and the data structures used by DNOS in controlling bidding, loading, and synchronizing tasks.

10 I/O Subsystem -- Reviews the overall I/O processing structure and the details of handling of device I/O, I/O utility calls, interprocess communication (IPC), file security, and name management.

11 Disk Structures and File I/O -- Includes an overview of file management, a description of disk and in-memory file structures, and a detailed description of key indexed file management.

12 DNOS System Tasks -- Discusses the conventions used in writing DNOS system tasks and provides detailed descriptions of many system tasks provided with DNOS.

13 System Generation Utility -- Provides an overview of the system generation utility and the data structures used.

14 Logging and Accounting -- Describes the functions and the flow of control for the system log and job accounting functions.

15 DNOS Performance Package -- Discusses the conventions used in system source code to enable the performance package and describes the routines executed in microcode.

16 DNOS Development and Analysis Tools -- Describes several tools available to Texas Instruments internal users for development purposes only and several tools and command procedures available for general access.

17 Analyzing a System Crash -- Describes the ANALZ utility functions and how to use them in analyzing a system crash file or in studying a running system.

18 XOP Processing -- Describes the XOP processors in DNOS and how to add a new XOP processor.

19 Special SVCs -- Describes SVCs used only by the operating system.

20 Linking Information for DNOS -- Explains how to link DNOS and provides examples of link streams and link maps from building a DSR link, a link of a system task, and the link of the DNOS root.

21 DNOS Source Disk Structure -- Describes the directories and files provided with a DNOS source kit.

22 Data Structure Pictures -- Provides data structure pictures for DNOS data structures commonly needed to understand the system.

A Keycap Cross-Reference - Discusses the generic keycap names that apply to all terminals that are used for keys on keyboards through out this manual.

For further information related to the use of DNOS, refer to the following document and those shown in the frontispiece.

Title	Part Number
DNOS Source Installation Guide	2270515-9701

TABLE of CONTENTS

Paragraph Title Page

SECTION 1 HOW TO USE THE DESIGN DOCUMENT

SECTION 2 OVERVIEW OF DNOS

SECTION 3 NAMING AND CODING CONVENTIONS

SECTION 4 DNOS STRUCTURE AND NUCLEUS FUNCTIONS

SECTION 5 IPL AND SYSTEM LOADERS

SECTION 6 SVC REQUEST PROCESSING

SECTION 7 SEGMENT MANAGEMENT

SECTION 8 JOB MANAGEMENT

SECTION 9 PROGRAM MANAGEMENT

SECTION 10 I/O SUBSYSTEM

SECTION 11 DISK STRUCTURES AND FILE I/O

SECTION 12 DNOS SYSTEM TASKS

SECTION 13 SYSTEM GENERATION UTILITY

SECTION 14 LOGGING AND ACCOUNTING

SECTION 15 DNOS PERFORMANCE PACKAGE

SECTION 16 DEVELOPMENT AND ANALYSIS TOOLS

SECTION 17 ANALYZING A SYSTEM CRASH

SECTION 18 INTERRUPTS AND XOP PROCESSING

SECTION 19 SPECIAL SVCs

SECTION 20 LINKING INFORMATION FOR DNOS

SECTION 21 DNOS SOURCE DISK STRUCTURE

SECTION 22 DATA STRUCTURE PICTURES

APPENDIX A KEYCAP CROSS-REFERENCE

LIST of FIGURES

LIST of TABLES

SECTION 1

HOW TO USE THE DESIGN DOCUMENT

The description of DNOS design is divided into sections according to major operating system functions. The nucleus routines are described first, along with their data structures and the overall operating system structure. This section is followed by separate sections describing each of the major subsystems in DNOS. For an overview of all subsystems, skim through this document, reading carefully the overview portion of each subsystem section. For details on a particular subsystem or module within a subsystem, consult the detailed diagrams and discussion that follow the overview.

Section 3 details naming conventions for the DNOS modules. When searching for details about a particular module, use the module name to determine which subsystem description is relevant. For details about particular data structures being used, consult the section on data structure pictures.

The section on linking information provides example link control streams used to build pieces of the operating system. To build a device service routine (DSR) or a new system task, use these link examples as a guide in building the required link streams. Link streams are also shown for several other parts of the operating system to show how these pieces are structured. The DNOS link streams should be considered the primary source of information about the modules included to support a particular task or subsystem. The link streams are also the primary source for full pathnames for modules in DNOS.

Most data structure pictures in this document are built directly from the templates copied into operating system source code. The structures are shown with hexadecimal byte counts, special comments, flags, and a diagram.

Most of the special terms used to describe DNOS can be found in the glossary in the DNOS Concepts and Facilities manual. Other terms are defined in this document as they are needed. Acronyms for system structures and routine names are introduced at various points throughout the manual. If you read a section from the manual without reading all preceding sections, an acronym may be encountered without an explanation of its meaning. Table 1-1 lists most of the acronyms used in the manual. Refer to this list in conjunction with the glossary for a complete description of the term.

Table 1-1 Acronyms Used in this Manual

Acronym -------	Meaning -------
ACC	Accounting record contents
ADR	Alias descriptor record
ADU	Allocatable disk unit
BRB	Buffered request block
BRO	Buffered request overhead
BTA	Buffer table area
BTB	B-tree block
CCB	Channel control block
CDE	Command definition entry
CDR	Channel descriptor record
CDT	Command definition table
DIA	Diagnostic status
DIB	Device information block
DOR	Directory overhead record
DPD	Disk PDT extension data
DPR	DIOU Device Parameters
DSR	Device Service Routine
FCB	File control block
FDB	File directory block
FDR	File descriptor record
FID	File identification
FIR	File information record
FMT	File management task area
FSC	File structure common
IOU	I/O utility task
IPC	Interprocess Communication
IRB	I/O request block
JCA	Job communication area
JIT	Job information table
JMR	Job manager request block
JSB	Job status block
KCB	KIF currency block
KDB	KIF descriptor block
KDR	Key descriptor record
KIB	KIF information block
KIT	KIF task area
KSB	Keyboard status block
LDT	Logical device table
LFD	Log file definition
LPD	Line printer PDT extension
LSE	Load segment entry
MRB	Master Read/Master Write buffer
NDB	Name definition block
NDS	Name definition segment

Table 1-1 Acronyms Used in this Manual (Continued)

Acronym	Meaning
NRB	Name manager request block
OAD	Overlay area description
OAW	Overlay area wait block
OSE	Owned Segment Entry
OVB	Overhead beet
OVT	Overlay table entry
PBM	Partial bit map
PDT	Physical device table
PFI	Program file directory index
PFZ	Program file record zero
PRM	DIOU/IOU Parameters
QHR	Queue header
RDB	Request definition block
RIB	Return information block
RLT	Record lock table
ROB	Resource ownership block
ROM	Read-only memory
RPB	Resource privilege block
RST	Reserve segment table
SAT	Secondary allocation table
SCO	Track 0, sector 0 format
SCI	System Command Interpreter
SDB	Stage descriptor block
SGB	Segment group block
SLB	System log block formats
SMT	Segment manager table
SOB	Segment Owner block
SSB	Segment status block
STA	System table area
STE	Swap table entry
TDL	Time delay list entry
TOL	Time-ordered list
TPCS	TILINE peripheral control space
TSB	Task status block
UDR	User descriptor record
WCS	Writable control store
WOM	Waiting for memory queue
XTK	Extension for a terminal with keyboard

SECTION 2

OVERVIEW OF DNOS

2.1 INTRODUCTION

DNOS, a general purpose operating system for the 990 computer, is designed to meet a variety of computing needs. DNOS is a configurable operating system, allowing users to generate small systems with minimal software development capability; medium-range systems with a limited number of options; and large systems with a wide variety of system options.

Among the special features available for DNOS are program and overlay loading, program swapping, key indexed files, dynamic job creation, output spooling, dynamic system configuration, interprocess communication (IPC), multiprogramming support, file access security, and a wide variety of utilities.

A performance package is available for DNOS. It uses microcode implementations of a number of DNOS routines to enhance the processing speed of DNOS.

2.2 GENERAL STRUCTURE

DNOS is composed of memory-resident and disk-resident code. The memory-resident portion includes the following:

* Device service routines

* Interrupt processors

* Extended operation (XOP) processors

* System tables and device buffers

* Many supervisor call (SVC) processors

* Task scheduler

* Nucleus support functions

* Memory-resident tasks

These parts are linked when DNOS is generated, and they are loaded into memory during initial program load (IPL). This memory-resident portion is referred to as the kernel of DNOS. The first portion of the kernel is referred to as the root; it forms the first segment of the mapping structure for kernel activities and for system tasks.

Disk-resident parts of DNOS include system tasks and overlays for some system tasks. These tasks include the I/O Utility, the Job Manager, and a number of miscellaneous SVC processors. These tasks are loaded into memory whenever their services are required.

Most of the DNOS functions are performed by routines that serve queues of requests. A queue is a first-in, first-out list of data to be processed. Each queue consists of a queue anchor, from which blocks of data are linked. Most queue anchors are located in the system root; those for file management are in the job communication area of the user job. The queues are singly linked, and the anchor points to the first data block, the first block points to the second, and so on. The anchor also points to the last block in the queue to enable efficient queue handling. The queue header format is displayed in the section of data structure pictures as the QHR.

2.3 FLOW OF CONTROL OF DNOS

While DNOS is running user jobs, the control paths vary. The diagram in Figure 2-1 shows an overview of DNOS initialization, functioning, and termination. Detailed paths for the various subsystems are described in the following sections.

```
                 Halt-Load Sequence
                 on 990 Front Panel
                         |
                         V
                Initial Program Load
                   (loads kernel and memory-resident
                    tasks, installs disk volumes, etc.)
                         |
                         V
                System Restart Task
                   (sets up system log and accounting log,
                    bids required system tasks, etc.)
                         |
                         |<************************************+
                         V                                     |
               Serve a Request of the                          |
                 Functioning System                            |
                         |                                     |
                         V                                     |
        +************************************+                 |
******> |           |        |               |                 |
|       V           V        V               V                 |
|    Execute     Process   Process    Provide Operating System |
|     Task        SVC     Hardware   Support for Memory       |
|     Code       Request Interrupt   Management, Timing,      |
|       |           |       or       Performance, etc.        |
|       \           /       XOP            |                   |
|        \         /         |             |                   |
|        Time Slice          |             |                   |
|          Expired           |             |                   |
|             ?              V             V                   |
<************* ******>***************>*******************>*****+
    no          yes        |
                           ~
                          |
                     error path only
                          |
                           ~
                           | Forced error condition, error in
                           | a system task, error in DNOS, or
                           | error in hardware
                           V
                   DNOS Crash Routine
                           |
                           V
                       System Halt
```

Figure 2-1 Flow of Control in DNOS

2.4 DX10 COMPATIBILITY

For a number of users, DNOS is an upgrade from the DX10 operating system. Most software that executes under DX10 executes without source change under DNOS. It needs only to be relinked with DNOS run-time support. The notable exceptions are user-written DSRs, XOP processors, system tasks and utilities, and SVC processors. Several sections of this document describe the changes needed to make these pieces of software function under DNOS.

Several system SVCs that were used with DX10 are not available for use in DNOS; in most cases, their functions are performed by a new SVC.

SECTION 3

NAMING AND CODING CONVENTIONS

3.1 NAMES OF ROUTINES

DNOS modules are written in either assembly language or Pascal. In most cases, a module consists of one routine. When several small routines perform related functions, those routines appear in a single module. Each routine and module is named using the form aabbbb where aa is an abbreviation for the subsystem in which the routine fits and bbbb is a set of characters that describe the function of the routine. For example, JMHALT is the job management routine that processes the Halt Job SVC.

Abbreviations for subsystems in the DNOS kernel and major utilities are shown in Table 3-1.

Modules are organized into directories that correspond to DNOS subsystems. Table 3-2 lists the major directories that comprise DNOS and indicates the section in which each directory is described. Other directories include modules for the various utilities of DNOS. The major directories labeled DNOS are detailed in this document; those labeled SCI, UTILITIES are detailed in the <u>DNOS SCI and Utilities Design Document</u>.

The source library for DNOS has one or more of the following subdirectories for each of the directories:

- PSOURCE for Pascal source
- FSOURCE for Fortran source
- SOURCE for Assembly language source
- MSOURCE for /12 microcode
- MOBJECT for assembled microcode
- MLIST for Microcode assembly listing
- TSOURCE for Link Editor modules needing to be transliterated from POPs code to assembly language
- UTILITY for the transliteration utilities for linker code

Table 3-1 DNOS Subsystem Abbreviations

Abbreviation	Subsystem or Utility
D$	Debugger
DM	Disk management
DS	Device service routines (DSRs)
DU	Device I/O Utility
E$	Text Editor
FM	File management
IO	I/O routines
IP	Interprocess communication (IPC)
IU	I/O utilities
JM	Job management
KM	Key indexed file (KIF) management
LG	System log and accounting log
MB	Mailbox
NF	Nucleus functions
NM	Name management
OI	Operator Interface
PL	Pascal-to-assembly-language interface
PM	Program and memory management
RP	Request processing - SVC support
SE	Security
SL	System loaders
SM	Segment management
SO	System overlay management
SP	Output spooler
TP	Teleprinter device utilities
UT	Subroutines common to several utilities
aaa	SCI utilities - aa or aaa is the SCI command

Table 3-2 Major Directories of DNOS

Directory	Location of Documentation
ANALZ	DNOS - Section 17
BATCH	DNOS - Section 21
DEBUG	DNOS - Section 16
DEBUGGER	SCI, UTILITIES
DEVDSR	DNOS - Section 10
DIOU	DNOS - Section 10
DISKMGR	DNOS - Section 12
EDITOR	SCI, UTILITIES
FILEMGR	DNOS - Section 11
IOMGR	DNOS - Section 10
IOU	DNOS - Section 10
IPC	DNOS - Section 10
JOBMGR	DNOS - Section 8
KIFMGR	DNOS - Section 11
LINK	DNOS - Section 20
LOADERS	DNOS - Section 5
LOG	DNOS - Section 14
LOGON	DNOS - Section 12
MACROS	DNOS - Section 3
MAILBOX	SCI, UTILITIES
MESSAGES	SCI, UTILITIES
NAMMGR	DNOS - Section 10
NUCLEUS	DNOS - Section 4
OPERATOR	SCI, UTILITIES
PASASM	DNOS - Section 3
PERFORM	DNOS - Section 15
PROGMGR	DNOS - Section 9
REQPROC	DNOS - Section 6
RESTART	DNOS - Section 12
S$	SCI, UTILITIES
SCI990	SCI, UTILITIES
SECURITY	DNOS - Section 10
SEGMGR	DNOS - Section 7
SPOOLER	SCI, UTILITIES
SYSJEN	DNOS - Section 13
SYSOVLY	DNOS - Section 12
TEMPLATE	DNOS - Section 3,22
TIGRESS	DNOS - Section 16
TPCALANS	SCI, UTILITIES
UTCOMN	SCI, UTILITIES

3.2 GLOBAL DATA AND STRUCTURE TEMPLATES

Names of system tables and data structures are generally three characters long, with the characters chosen to reflect the structure name. Fields within the structure have six-character names (whether part of Pascal records or assembly language code); the first three characters are the same as the structure label. Flag fields within the structure are detailed using equates, with each flag bit (or set of bits) identified by aaFbbb where aa represents the first two characters of the structure name, F indicates a flag, and bbb describes the flag. For example, the task status block is the TSB. TSBPRI is the name a field within the TSB that carries the task priority. TSFSYS names a flag in the TSB that indicates whether a task is a system task.

Global constants and error equates are named using these formats:

```
WDaaaa   DATA  >aaaa
ERRaa    BYTE  >aa
BYTEaa   EQU   ERRaa
```

where a is a hexadecimal digit and > is used to represent a hexadecimal value.

The TEMPLATE directory contains the global constants and variables used by DNOS source code. In the following list of subdirectories of the TEMPLATE directory, DSC is a synonym for the entire DNOS source directory. All of these directories, except the PREAMBLE directory, also appear in the linkable parts directory .S$OSLINK on an installed DNOS system.

```
DSC.TEMPLATE.ATABLE
DSC.TEMPLATE.COMMON
DSC.TEMPLATE.DECLARE
DSC.TEMPLATE.PREAMBLE
DSC.TEMPLATE.PTABLE
```

The DSC.TEMPLATE.ATABLE directory contains templates for DNOS data structures that are used by assembly language routines. Files in this directory are copied into an assembly language module to reference fields within the data structures. A module accesses a particular field in a structure by using a template offset with a pointer. The pointer can be passed to the module or retrieved from some other DNOS structure. This directory also includes a template of system crash codes (NFCRSH) and a template of task states (NFSTAT). Files from this directory are shown in detail in the section on data structure pictures. They are built using the picture macros described in the section on DNOS Development and Analysis Tools.

The DSC.TEMPLATE.COMMON directory contains the common data used by assembly language routines. It consists of files of CSEG blocks, including the following major files:

- DSC.TEMPLATE.COMMON.NFDATA - Global data values for the current state of the system

- DSC.TEMPLATE.COMMON.NFERn0 - Byte constants (>n0 through >nF) and equates for system error codes (16 such templates)

- DSC.TEMPLATE.COMMON.NFPTR - System pointers to global lists and structures

- DSC.TEMPLATE.COMMON.NFWORD - Word constants

Any assembly language module that makes use of a byte constant or a word constant copies the appropriate common template and uses the constant in that module. Similarly, NFPTR and NFDATA are copied into a module to allow access to a system pointer or global data item. Use of the templates provides documentation of all uses of a particular error code, constant, or system variable.

NFPTR includes pointers to system queues, pointers to beginnings of structure lists, addresses of segment management tables, pointers to device information, and several miscellaneous pointers. Full details on NFPTR appear in the section on data structure pictures.

NFDATA includes anchors for several system data structures, counts for jobs and tasks in the system, parameters for system time units, sizes of the system and system files, scheduling data, a word of flags which define system options chosen at system generation, and several other items. Details of NFDATA are shown in the section on data structure pictures.

The DSC.TEMPLATE.DECLARE directory is used by Pascal routines. It consists of files of procedure declarations, which are copied into Pascal modules. Each subsystem or utility written in Pascal has a file of declarations for its own set of modules. Also, declarations are included for run-time routines and for interface routines from Pascal code to assembly language modules.

The DSC.TEMPLATE.PREAMBLE directory has the templates for documentation preambles to assembly language and Pascal source modules.

The DSC.TEMPLATE.PTABLE directory has data structure templates and common segment templates for use by Pascal code. The directory includes files corresponding to each of those in the DSC.TEMPLATE.ATABLE and DSC.TEMPLATE.COMMON directories. Also, it has a number of files that have no counterparts in the other directories but are needed by Pascal routines.

3.3 ASSEMBLY LANGUAGE CODING CONVENTIONS

Each assembly language module begins with a preamble that describes that module. Fields in the preamble template that are not used for a particular module are omitted in that module. In the assembler template, the following are required sections:

copyright statement
routine name
abstract
entry
exit
errors
revision
environment
IDT name
PSEG, code, and END

When more than one routine is included in a module, each routine is preceded by a description that includes abstract, entry, exit, and error information.

The abstract gives a brief English description of the general purpose of the module, while the algorithm section describes how the routine works. The environment section points out what table areas are used by the module. Revision information is provided in the format shown below. Other entries are self-explanatory.

```
* REVISION: <creation date mm/dd/yy - ORIGINAL
* <revision ID> - <date> - <purpose> - <OS release no>
*      repeated, with latest revision last
```

where:

<revision ID> is a pair of decimal digits, beginning with 01.
<date> is the form mm/dd/yy, where each field is decimal.
<purpose> is description of the change, including the number of any STR on design request being satisfied by this revision.
<OS release no> is the release for which this revision was prepared.

To keep track of which lines of code were added for what revisions, each added line is flagged. In columns 58 through 60 of the line added to an assembly language module, the characters Rmn are inserted, where <mn> is the revision ID specified for this revision in the preamble.

Templates copied into assembly language programs with the COPY statement are by default UNListed. (Data templates and other structures are surrounded by UNL and LIST. To see the copied items, the program may be assembled with the FUNLST (F) option of the assembler enabled.)

For the most part, assembly language code uses tab settings of 1, 8, 13, 31, and a right margin of 60 to make the assembly listing as legible as possible. Comments are included in the preamble and in atoms within each routine. An atom is several lines of comments, set off from the code it describes.

Labels used within an assembly language routine are composed of three characters followed by three digits (for example, OPN100 for a label in a routine performing open processing). The characters are chosen from the routine name unless another set of characters is clearly more useful. The numeric portion ends in zero to allow room for inserted labels, and labels appear in ascending numeric order from beginning to end of a module.

The format of the assembly language preamble is as follows:

```
       TITL  '<MODULE ID - SHORT DESCRIPTION>'
*
*   (C) COPYRIGHT, TEXAS INSTRUMENTS INCORPORATED, 1983.
*    ALL RIGHTS RESERVED.  PROPERTY OF TEXAS INSTRUMENTS
*    INCORPORATED.   RESTRICTED RIGHTS - USE, DUPLICATION
*    OR DISCLOSURE IS SUBJECT TO RESTRICTIONS SET FORTH
*    IN TI'S PROGRAM LICENSE AGREEMENT AND ASSOCIATED
*    DOCUMENTATION.
*
* ROUTINE NAME: <NAME OF ROUTINE(S)>
*
* ABSTRACT: <DESCRIBE THE GENERAL PURPOSE OF THIS ROUTINE>
*
* ENTRY:     <INSTRUCTION/STATEMENT/INTERRUPT USED TO ENTER>
*            (<RN>) = <DESCRIPTION>
*
* EXIT:      (<RN>) = <VALUE> IF <CONDITION>
*
* ERRORS:    <ACTION OR CODE> IF <CONDITION>
*
* STACK REQUIREMENTS: <N> WORDS
*
* ALGORITHM:   <DESCRIPTION OF ALGORITHM IF NECESSARY>
*
* REVISION: <CREATION DATE IN MM/DD/YY> - ORIGINAL
*           <REVISION DATE; LATEST LAST> - <NATURE>
*
* ENVIRONMENT: 990/10 ASSEMBLER
*              CALLABLE FROM <assembler,Pascal>
*              TABLE SEGMENTS MAPPED IN WHEN ENTERED:
*                             <LIST>
*              TABLE SEGMENTS MAPPED IN DURING ROUTINE:
*                             <LIST>
*
* NOTES:     <SPECIAL CONDITIONS/ASSUMPTIONS OR OTHER
*            SPECIAL INFORMATION>
*
*
* SUBROUTINE REFS:
       REF  <NAME>             <DESCRIPTION>
*
* CONDITIONAL ASSEMBLY:
*      <VARIABLE>              <DESCRIPTION>
*
* MACROS TO BE USED:
       LIBIN DSC.MACROS.TEMPLATE
*      LIBIN DSC.<MACRO LIBRARY PATHNAME>
*
* EQUATES:
<NAME> EQU  <VALUE>            <DESCRIPTION>
* <INSERT COPIES OF EQUATE FILES IF ANY>
*
* GLOBAL DATA  (TO SHARE AND ACCESS DATA IN COMMON AREAS)
```

```
*
* <INSERT COPIES OF ANY RELEVANT CSEG FILES>
       PAGE
*
       IDT '<MODULE NAME>'
       DEF  <MODULE NAME>
       PSEG
<ROUTINE CODE>
       END
```

Several macro libraries are available in the MACROS directory for use by assembly language routines. In each case, the SOURCE file shows the macro definitions and documents their use. To find out how a particular macro functions, read the comments in the source file for the macro.

The DSC.MACROS.TEMPLATE library must be included with a LIBIN statement in most modules that use data structure templates. Many of the templates in the DSC.TEMPLATE.ATABLE directory are defined using macros that allow processing in assembly language and Pascal structures. It includes macros for ADDR, BITS, CHAR, FLAG, FLAGS, LONG, PTR, RECORD, WORD, INT, PCKREC, ENDREC, REC, ARRAY, POSINT, and VARNT.

In the rare instance that a CSEG must be used as a DSEG, the set of macros in DSC.MACROS.DORGCSEG should be used. This set includes macros for CSEG, CEND, and DZERO directives. These macros are often used by modules that issue the Retrieve System Data SVC (>3F) to access a part of a system common area. The SVC expects the user to specify an offset into the common area (as a DSEG would allow) rather than an absolute CSEG address.

The DSC.MACROS.FUNC library includes macros to inhibit and enable scheduling, to initialize a block of data, to test conditions during assembly of code, and to provide common subroutine access. This set includes macros for ASSUME, DATAM, ENAB, INHB, SCALL, SPOP, SPUSH, GTA, GTAO, RTA, PRCK, SGCK, SRTN, and TRTN. These macros must be used for the purposes described in Table 3-3. See the FILE DSC.MACROS.FUNC.SOURCE for the descriptions of the macro details. All accesses to the routines indicated in Table 3-3 must be made using the macros, since the macros provide access to performance microcode,

Table 3-3 Macros from DSC.MACROS.FUNC

MACRO NAME	PURPOSE
ASSUME	Test an assembly condition, (generally a template field)
DATAM	Generate data fields
DCLOSE	Door Close
DOPEN	Door Open
ENAB	Access NFENAB to enable scheduling
INHB	Inhibit scheduling
SCALL	Call another routine
SPOP	Access NFPOP
SPUSH	Access NFPSH
GTA	Access NFGTA
GTAO	Access NFGTAO
PRCK	Access RPPRCK
SGCK	Access RPSGCK
RTA	Access NFRTA
SRTN	Access NFSRTN
TRTN	Access NFTRTN

Macros in DSC.MACROS.UTILITY are used by a number of DNOS and SCI utility programs to perform commonly needed operations. It includes macros to terminate a program under abnormal error conditions and a variety of special field initialization macros.

A set of macros is available to build assembly language routines to be called by Pascal routines. These macros yield code compatible with Pascal subroutine conventions. The macros are in the library named DSC.MACROS.RIFLE.MACROS.

3.4 PASCAL CODING CONVENTIONS

Several subsystems are written in a subset of TI Pascal. These include job management, system generation (sysgen), system log processing, accounting log processing, and many SCI utilities.

Statements are written one per line, and segments of programs are visibly separated to facilitate readability. As with assembly language programs, Pascal programs are documented in the preamble and throughout the code. To allow printing of source code on any available printer, only uppercase characters are used.

In the Pascal template, the following fields are required:

compiler options
copyright statement
program statement
abstract
revision
environment
procedure (function) and code

The revision information must be of the following format:

```
   " REVISION: <creation date mm/dd/yy - ORIGINAL>
   " <revision ID> - <date> - <purpose> - <OS release no>
   "      repeated, with latest revision last
```

To keep track of which lines of code were added for what revisions, each added line is flagged. For Pascal code, the characters Rmn are inserted with a comment indicator after column 60.

The preamble template for a Pascal module is of the following form:

```
(*&FILL-,ADJT-,SLIM(72)*)
"
"     (C) COPYRIGHT, TEXAS INSTRUMENTS INCORPORATED, 1983.
"     ALL RIGHTS RESERVED.  PROPERTY OF TEXAS INSTRUMENTS
"     INCORPORATED.  RESTRICTED RIGHTS - USE, DUPLICATION
"     OR DISCLOSURE IS SUBJECT TO RESTRICTIONS SET FORTH IN
"     TI'S PROGRAM LICENSE AGREEMENT AND ASSOCIATED
"     DOCUMENTATION.
"
"
(*$WIDELIST,NO MAP,LOCALS,GLOBALS*)
PROGRAM     <DUMMY NAME>;
"
" ROUTINE NAME: <NAME OF ROUTINE(S)>
"
" ABSTRACT  <DESCRIBE THE GENERAL PURPOSE OF THE ROUTINE>
"
" NOTES:    <SPECIAL CONDITIONS, ASSUMPTIONS, OR OTHER SPECIAL
"           INFORMATION>
"
" METHOD:  <DESCRIPTION OF ALGORITHM IF NECESSARY>
"
" REVISIONS: ORIGINAL <MM/DD/YY>;
"            REVISION <INTEGER>: <MM/DD/YY>, <PURPOSE OF REVISION>
"
" ENVIRONMENT: 990/10 PASCAL X.X
"              TABLE SEGMENTS MAPPED IN WHEN ENTERED
"                   <STA, JCA OR OTHER TABLE>
"              TABLE SEGMENTS MAPPED IN DURING ROUTINE
"                   <STA, JCA OR OTHER TABLE>
(*$PAGE*)
"
"                         GLOBAL  DECLARATIONS
```

```
"
CONST       <IDENTIFIER> = <CONSTANT EXPRESSION>;   (*<DESCRIPTION>*)
?COPY <FILENAME OF GLOBAL CONSTANTS>;
TYPE        <IDENTIFIER> = <TYPE>;                  (*<DESCRIPTION>*)
?COPY <FILENAME OF GLOBAL TYPES>;
COMMON      <IDENTIFIER> : <TYPE>;                  (*<DESCRIPTION>*)
?COPY <FILENAME OF COMMONS>;
ACCESS      <IDENTIFIER>,
            <IDENTIFIER>;
"
"      FUNCTIONS OR PROCEDURES DEFINED EXTERNAL TO THIS MODULE
"
<INSERT THE ?COPY THAT BRINGS IN PROCEDURE DECLARATIONS FOR THIS
 SUBSYSTEM, WHERE EACH PROCEDURE IS DEFINED WITH ITS PARAMETERS AND
 DECLARED AS BEING FORWARD>;
"
(*$PAGE*)
"
PROCEDURE   <PROCEDURE NAME)>;
"<COMMENT HERE THE PROCEDURE NAME WITH ITS PARAMETERS AS A READING AID>
"
"                      LOCAL DECLARATIONS
"
LABEL       <INTEGER>,       (*<DESCRIPTION>*)
            <INTEGER>,       (*<DESCRIPTION>*)
CONST       <IDENTIFIER> = <CONSTANT EXPRESSION>;   (*<DESCRIPTION>*)
            <IDENTIFIER> = <CONSTANT EXPRESSION>;   (*<DESCRIPTION>*)
TYPE        <IDENTIFIER> = <TYPE>;
            <IDENTIFIER> = <TYPE>;
VAR         <IDENTIFIER>, <IDENTIFIER>  : <TYPE>;
            <IDENTIFIER>, <IDENTIFIER>  : <TYPE>;
COMMON      <IDENTIFIER>,  <TYPE>;                  (*<DESCRIPTION>*)
            <IDENTIFIER>,  <TYPE>;                  (*<DESCRIPTION>*)
ACCESS      <IDENTIFIER>,
            <IDENTIFIER>,
"
"
BEGIN  (*$MAP*)
"
-*INSERT PROCEDURE CODE*-
"
END;
"
BEGIN  (*$NO OBJECT*)
END.
```

Pascal routines make use of the templates in the DSC.TEMPLATE.PTABLE directory through use of ?COPY statements. The data structure templates are copied in as type declarations, and the CSEG template equivalents are copied as common declarations. In addition to templates for DNOS structures, the DSC.TEMPLATE.PTABLE directory also includes a standard set of types for DNOS in DSC.TEMPLATE.PTABLE.TYPES.

The PASASM directory includes interface routines written in the Pascal.MACROS to allow routines written in Pascal to call DNOS kernel routines written in assembly language. Routine names begin with the letters PL and have the same last four characters as the nucleus routine to which they interface.

For utilities written in Pascal, a collection of routines is available for interface to SCI. These routines are like the S$ routines used by assembly language and are found in the Pascal object directory.

3.5 ERROR HANDLING

Errors detected by assembly language routines are encoded using error code constants and equated symbols from the collection defined in these copy modules:

DSC.TEMPLATE.ATABLE.NFCRSH	(for system crash codes)
DSC.TEMPLATE.COMMON.NFER00	(error codes >00 through >0F)
DSC.TEMPLATE.COMMON.NFER10	(error codes >10 through >1F)
DSC.TEMPLATE.COMMON.NFER20	(error codes >20 through >2F)
DSC.TEMPLATE.COMMON.NFER30	(error codes >30 through >3F)
DSC.TEMPLATE.COMMON.NFER40	(error codes >40 through >4F)
DSC.TEMPLATE.COMMON.NFER50	(error codes >50 through >5F)
DSC.TEMPLATE.COMMON.NFER60	(error codes >60 through >6F)
DSC.TEMPLATE.COMMON.NFER70	(error codes >70 through >7F)
DSC.TEMPLATE.COMMON.NFER80	(error codes >80 through >8F)
DSC.TEMPLATE.COMMON.NFER90	(error codes >90 through >9F)
DSC.TEMPLATE.COMMON.NFERA0	(error codes >A0 through >AF)
DSC.TEMPLATE.COMMON.NFERB0	(error codes >B0 through >BF)
DSC.TEMPLATE.COMMON.NFERC0	(error codes >C0 through >CF)
DSC.TEMPLATE.COMMON.NFERD0	(error codes >D0 through >DF)
DSC.TEMPLATE.COMMON.NFERE0	(error codes >E0 through >EF)
DSC.TEMPLATE.COMMON.NFERF0	(error codes >F0 through >FF)

All SVC error codes and crash codes are documented in these copy modules. Errors detected by Pascal routines use the same codes, defining constants to have the appropriate error number. The meaning of each error code is described in detail in the DNOS Messages and Codes Reference Manual. These errors are also viewable with the Show Expanded Message (SEM) command.

3.6 GENERATING NEW ERROR CODES

The current set of error codes must be very carefully examined when a new error code is added. For SVCs, the new error code must not duplicate any previously defined code which might arise for that SVC. The DSC.TEMPLATE.COMMON.NFERxx files and the SVC code list in the DNOS Messages and Codes Reference Manual must be examined. In addition to codes listed explicitly, an SVC processor may return an error code defined for I/O SVC 00 if an I/O SVC is executed by the processor. Thus, for SVCs in the following set, any new error code cannot duplicate any defined for SVC 00: >14, >1F, >20, >22, >25, >26, >27, >28, >29, >2A, >2B, >31, >34, >37, >38, >40, >43 and >48. The error codes reserved for I/O are annotated in the NFERxx files as being used for SVC 00, IOU, FILEMGR, DIOU, or IPC. Also, SVC >40 and SVC >43 must not share an error code, since SVC >43 returns errors from SVC >40.

Once a new error code has been chosen for an SVC error, that code must be documented in the appropriate NFERxx file. It also must be documented in the SVC files used by the error processing utilities. These files are DSC.MESSAGES.TEXT.SVC and DSC.MESSAGES.EXPTEXT.SVC. If the message employs variable text pulled from the offending call block, the appropriate entries must also be made to the tables in DSC.REQPROC.SOURCE.RPRCDA for use by the Return Code Processor SVC. The section on system tasks includes a description of the task RPRCP and its required data structures for handling the Return Code Processor SVC.

When adding error codes for non-SVC purposes, several sources must be examined. Task errors are documented in the NFERxx files as well as in the DNOS Messages and Codes Reference Manual. Any additions to the set must not duplicate previously defined codes, and appropriate updates must be made to the NFERxx files and the manual.

Additional system crash codes must be checked with the file DSC.TEMPLATE.ATABLE.NFCRSH and with the DNOS Messages and Codes Reference Manual. Additional error codes for SCI or utilities must be checked against currently defined codes as documented in the DNOS Messages and Codes Reference Manual. Further information about assigning error codes for SCI or utilities can be found in the DNOS SCI and Utilities Design Document.

SECTION 4

DNOS STRUCTURE AND NUCLEUS FUNCTIONS

4.1 OVERVIEW

DNOS uses the memory mapping option of the 990/10, 990/10A and 990/12 to efficiently divide the operating system code. It uses a number of common data structures and a set of system files to facilitate communication between subsystems. The DNOS nucleus includes the code for miscellaneous support functions, task scheduling and execution, interrupt processing, task termination, and SVC processing.

4.2 SYSTEM MEMORY MAPPING

Parts of DNOS run in map file 0, some parts run in map file 1 and other parts alternate use of each map file. Each map file is divided into three segments that may total up to 64K bytes of physical memory.

Task code is executed in map file 1. SVC support, device service routines (DSRs), interrupt support, and scheduling code are executed in map file 0. Several nucleus support routines may execute in either map file (depending on which is in use by the caller). Figure 4-1 shows the arrangements used by DNOS.

Map file 0 contains the following: sl

* First map segment (system root):

  - Interrupt and XOP vectors, interrupt decoder and tables

  - Nucleus common support routines

  - Common data segments

  - System table area (STA)

* Second map segment:

  - Job communication area (JCA) for the task currently executing, or

  - Special table areas as needed by subsystems, or
  - Buffers for I/O

* Third map segment:

  - Scheduler overlay including some SVC support, or
  - SVC support not included in scheduler overlay, or
  - DSR code as required by devices.
  - Task code running in a fast transfer mode

Map file 1 is set up in one of two ways, depending on whether or not the task is installed in a program file as a system task. If the task is a system task, the first map segment is set up the same as map file 0, the second map segment is set up with the JCA of the task, and the third map segment is set up with the task code. For nonsystem tasks, all three map segments may be used for task and procedure code (no system area is mapped into the task).

```
  First Segment           Second Segment            Third Segment

                        +------------------+      +------------------+
+------------------+    |       JCA        |      |Scheduler/SVC Code|  map 0
|   Kernel Code    |    +------------------+      +******************+
|******************|    +******************+      +******************+
|System Table Area|     |     SM Table     | 5,6  | SVC Code         |  map 0
+******************+    +******************+      +******************+
                        +******************+
                        |     FM Table     | 5,6
                        +******************+      +******************+
                        +******************+      |  DSR             |  map 0
                        | Synonym and Name| 1     +******************+
                        |     Segment      |
                        +******************+
                        | Physical Record  | 2    +******************+
                        |     buffer       |      | System Task Code |  map 1
                        +******************+      +******************+   or 0
                        +******************+
                        |   Disk Bit Map   | 3,5,6
                        +******************+
                        +******************+
                        |Device I/O Buffer| 4,5
                        +******************+
```

1 - mapped only by name management
2 - mapped only by file management
3 - mapped only by disk management
4 * mapped only by DSRs and I/O subsystem
5 * memory*resident
6 * fixed size

Figure 4*1 DNOS Map Files

4.3 SYSTEM DATA STRUCTURES

DNOS data structures include both common segments and dynamically allocated tables.

The two common segments that contain most of the system variables and pointers are NFDATA and NFPTR. The common segments containing most of the constants used in DNOS include: NFWORD, NFER00, NFER10, NFER20, NFER30, NFER40, NFER50, NFER60, NFER70, NFER80, NFER90, NFERA0, NFERB0, NFERC0, NFERD0, NFERE0, and NFERF0.

See the section on detailed data structures for more information.

The four areas from which system data structures may be allocated are as follows:

* STA, in the system root, where structures needed by more than one job are located

* JCAs, one for each job in the system, where job-local structures are located

* Segment Manager special table areas (see the section on segment management for details)

* File Manager special table areas (see the section on the I/O subsystem for details)

Each job in the system is represented by a job status block (JSB) in the STA. The JSB contains job identification information, links for various queues, and priority information.

Tasks (programs) executing in each job are represented by TSBs, kept in the JCA for the job in which the task is running. The TSB contains all of the information concerning the state of a task. This includes the current task status indicators of workspace pointer (WP), program counter (PC), and status register (ST); task state; task priority; flags; installed and run-time IDs; segment identifiers; map file registers; outstanding I/O counts; execution time; and end-action pointers for the WP and PC.

4.4 SYSTEM FILES

DNOS requires certain files to be on the system disk (primary disk) for its operation. These files are:

* The loader file, .S$IPL, containing the image of the IPL program (see the section on IPL and System Loaders)

* The kernel program file, containing the tasks, procedures, and overlays comprising DNOS

* The utilities program file, containing tasks and procedures for system utility programs

* The applications program file specified in SYSGEN, containing tasks and procedures for user programs

* The shared program file, .S$SHARED, on which users may place procedures to be shared by other program files, and where tasks and procedures are placed when installed to LUNO 0

* The swap file, .S$ROLLD.S$ROLLA, where task images are temporarily placed to make room in memory for higher*priority tasks

* The crash file, .S$CRASH, where an image of memory is written in the event of a system crash

Other files are also on the system disk for proper execution of SCI and various DNOS features. These files are:

* The command procedures directory of SCI commands, .S$CMDS

* The directory of command definition tables used to process keyboard bids, .S$CDT, with one file for each system booted on this disk

* The messages directories, .S$MSG and .S$EXPMSG. If these are not present, messages appear in cryptic form.

* The spooler queue directory, .S$SDTQUE, with one file for each system booted on this disk

* The system generation directory, .S$SGU$

* The overlay management directory, .S$SYSLIB

* A library of system programmer commands and the system history file in the directory .S$SYSTEM

* The user ID directory, .S$USER and the capabilities list file, .S$CLF

* Accounting files, .S$ACT1 and .S$ACT2, used when accounting is enabled

* The initialization batch stream .S$ISBTCH, used to start the Spooler and for user*specified activities

* System log files, .S$LOG1 and .S$LOG2

* The file .S$MVI, used by the Modify Volume Information processor to record changes to the disk

* The file .S$SCA, used by LOGON and SCI

* The program file .S$SECURE, used if file access security is generated with the system

File structures are described in detail in the section on file management.

4.5 NUCLEUS SUPPORT FUNCTIONS

The nucleus provides support routines for system tasks as well as for other parts of the nucleus. The routines support such things as routine linkage, queuing, synchronization, inhibiting scheduling, map file changes, table area management, and system crash analysis.

4.5.1 Linkage Support.

Most of the linkage between DNOS routines is accomplished by the push and pop routines (NFPSHn and NFPOPn, where n is the number of registers to push or pop). R10 is used throughout the DNOS code as a stack pointer. On entry to a routine, the return address is pushed on the stack, and a push routine is called to save registers on the stack. To exit from the routine, the return code is placed in the leftmost byte of R0 and a branch is made to the pop routine that corresponds to the push routine that was used. The assembly language macros SPUSH and SPOP must be used to set up the linkage to subroutines, since the performance microcode depends on their use. For example, the following code shows linkage using three registers:

```
          Entry:                      Exit:
          ******                      *****
          MOV    R11,*R10+            MOVB   @ERR30,R0
          BL     @NFPSH3              B      @NFPOP3
      or
          SPUSH  3                    SPOP   @ERR30
```

Most of the code in the kernel makes use of the stack defined in the scheduler segment. The scheduler stack is initialized at the NFSCHD and RPROOT entry points.

When the called routine makes use of SPOP to return to a caller, the calling routine can specify three types of error returns. The word following the BL instruction contains a return address to be used if an error occurs in the called routine. When the called routine branches to NFPOP, a test is made to see whether or not the leftmost byte of R0 is zero. If it is not zero, the return is made to the address specified for error handling. Two special cases can be specified as error addresses:

* 0 - indicates that there is no error possible or that the error should be ignored and, if one occurs, return to the same address that would have been used if no error had occurred

* -1 - indicates that no error return is expected and if one occurs a system crash (0029) should occur. (This case is primarily used during debugging of DNOS code.)

4.5.2 Queuing Support.

Many of the DNOS system tasks are queue servers, tasks dedicated to processing entries on queues. When an entry is placed on a queue server's queue, the queue server is activated (if it is not already active) and begins processing entries. When the processing is finished, the queue server either suspends and waits for more entries to be placed on the queue, or it terminates; depending upon the time-critical nature of the function being performed.

System data structures can be queued and dequeued to the following types of queues, using the nucleus queuing routines:

* Queues with one-word headers, whose entries form a singly linked list. The routines NFQUE1 and NFDQ1 are used to queue and dequeue the entries in a first-in, first-out manner.

* Queues with a six-word header, whose entries form a singly linked list. The header includes fields pointing to the first entry and to the last entry, and it contains a count of the entries. If the queue is being served by a queue server, the header also contains the task identifier for the queue server task as well as the TSB address, the JSB address, and the program file identifier. The routines NFQUEH and NFDQH are used to queue and dequeue entries in a first in, first out manner. NFQUEH activates the queue server when necessary.

* Queues of overhead beets (OVBs), whose entries form a doubly linked list. The routines for queuing and dequeuing overhead beets are NFLOVB and NFDLOV memory management lists and NFQOVB and NFDOVB for six-word headers.

Queue headers for system queues are maintained in two locations. Some queue servers execute in the system job and have their queue headers in the system root. Other queue servers execute in the user's job and have their queue headers in the user's job communication area (JCA). Queue headers are defined with an assembly language DEF directive for the header so that queue servers running in the system job can use an assembly language REF directive for the label and access the queue header address directly. Queue servers in the user's job receive the queue header address as their second task bid parameter and access the queue header using this address.

The form of a system queue header is shown in the queue header template, QHR. All system root queue headers are defined in the template, DSC.TEMPLATE.COMMON.NFQHDR. NFQHDR is copied during sysgen to initialize the queue headers. Some of the queues are optional, depending upon sysgen choices. If a queue is not used, the symbol for the queue header is defined as a word of zeros in the system root. The bid of a queue server is done by NFACTQ and the queue server terminates after processing the queue of requests.

During system start-up, to prevent premature request processing, the queue server IDs in several queue headers are temporarily set to zero. When the system is ready to handle the requests, the queue server ID is restored. These operations are done by RESTART.

Queue headers in the job communication area are built when a job is created. The queues for program file SVC operations (install, delete, assign space, map name to ID), Initialize New Volume SVC, and Return Code Processor SVC are in the user job communication area. The section on writing system tasks describes how to build tasks for each of these environments.

4.5.3 Synchronization and Coordination.

Some nucleus routines aid in coordinating access to the same system structure or code by more than one routine. One such coordination aid is the door. A door is described by a two-word descriptor record that is passed to the door-handling routines. The routine NFDCLO closes a door and prevents other tasks from accessing the door until it is opened by NFDOPN. A task trying to access a door that is closed is suspended until the door is opened. The macros DCLOS and DOPEN are used to call these routines. These macros are in DSC.MACROS.FUNC. (This type of coordination may also be accomplished by using the semaphore SVC. See the section on program management.)

4.5.4 Inhibiting Scheduling.

When a task is executing critical code, scheduling must not occur. One assembly language macro is used to inhibit the scheduler (INHB) and another to enable the scheduler (ENAB). Between the execution of INHB and ENAB, the task will not be rescheduled. These macros are located in DSC.MACROS.FUNC.

4.5.5 Map File Changing.

Occasionally, a system task (which normally executes in map file 1) must call a routine that can execute only in map file 0. Interface routines are available for switching to map file 0 upon entering a routine and returning to map file 1 upon exit. A routine is entered in map file 0 by executing a BLWP @NFMAPO, with the next word of program code specifying the address of the routine to be entered. The second word following the BLWP contains an error address, zero, or -1. If an error address is specified and an error occurs, the return from the called routine is made to the error address. If zero is specified and an error occurs, no special action is taken; execution continues. If -1 is specified and an error occurs, the system crashes with a crash code of >0029 (this is used primarily during debugging of DNOS). The called routine returns to the caller in map file 1 by branching to NFRTNO.

When a routine executes in map 0, it expects to be using the scheduler workspace. Thus it is necessary to set up any required registers in that workspace before calling NFMAPO. It is also necessary to pass back any data, (including any error code in RO) before calling NFRTNO.

4.5.6 Table Area Management.

The routines in module NFTMGR allocate and deallocate table area in the dynamic table areas. Allocation is performed by NFGTA and NFGTAO (initialized to zero after allocation), and deallocation is performed by NFRTA. The smallest block of table area allocated is eight bytes. When memory in the specified table area is exhausted, an error is returned to the caller. Macros GTA, GTAO, and RTA must be used to access these routines. These routines may not be called from code which processes interrupts or requires interrupts to be masked.

The Segment Manager support routines enable system functions to map special table areas, find segment status block (SSB) addresses for segments, create and delete SSBs and SGBs, and force load segments into memory. Descriptions of these routines follow:

SMMJCA, SMMJC1, and SMMJC2

Maps JCAs into the second segment of the executing task or processor map file. When called from map file 0, each of these routines performs the same processing, simply mapping the requested JCA into the current map file 0. When called from map file 1, SMMJCA does not change the releasable and modified status of the old segment. SMMJC1 allows the caller to specify the releasable and modified status. SMMJC2 is used if the caller needs an error code rather than loading of the JCA when the JCA is not in memory; otherwise SMMJC2 functions like SMMJCA.

SMMTBL and SMMTB1

Maps special table areas into the second segment of the executing task or processor map file. These two routines function like SMMJCA and SMMJC1.

SMMSEG

Maps an arbitrary segment into the second segment of the executing task map file. SMMSEG allows the caller to specify a byte offset which is to be the beginning of the mapped portion of the segment. Specifying an offset of zero causes the entire segment to be mapped.

SMCSGO

Maps an arbitrary segment into the second segment of the executing task map file. SMCSGO does the actual work of and is a common subroutine of SMMJCA, SMMTBL, and SMMSEG.

SMSRCH

Returns an SMT/SSB pair for a specified ID/file descriptor packet (FDP) pair. SMSRCH calls SMFSID to see if an SSB exists for the specified ID. If so, it verifies that the caller has access to the segment, which may include an SMCHUC call. If the caller has access, the SMT/SSB pair is returned. If not, SMSRCH will return a replicated SSB if the segment is replicatable; otherwise an error is returned. If no SSB already exists, SMBLDS is called to create one.

SMBLDS

Creates an SSB (and an SGB if necessary) for a given segment type. The caller specifies an FDP and a task/procedure flag. If the FDP is zero, a memory-based segment is built. SMBLDS first builds an SGB if there is none for the specified file. It then builds an SSB of the correct size, supplies a run-time ID, and links the SSB onto the SGB. For data files, the length and attributes are set; for program files, certain flags are set.

SMFSID

Searches a segment group for a segment with a specified ID. The caller specifies the segment group via an FDP address. If the FDP address is zero, the memory-based segment group is assumed. The caller can search for the segment via an installed or run-time ID. Also, the caller can search for a task segment. If a match is found, the Segment Manager table area that contains the segment's SSB is mapped. This routine is callable by system tasks and processors.

SMCHUC

Checks to see if the use counts of a given segment can all be accounted for by the mapped or loaded segments of a task.

SMLOAD

Loads a segment into memory for system tasks if the segment is not already in memory. The segment is not mapped into the task address space but remains in memory as long as the task is in memory. A segment may be loaded by more than one task, regardless of its attributes. The use count and task-in-memory count of the segment are incremented. This routine also serves the function of an SVC processor.

SMUNLD

Unloads a segment loaded by SMLOAD. SMUNLD detaches the segment from the task; consequently, the segment need not be in memory when the task is in memory. This routine decrements the use and task-in-memory counts for the segment. This routine also serves the function of an SVC processor.

SMDSSB

Deallocates segment memory and deletes a specified SSB. If the segment (specified by the SMT/SSB pair) is not used, reserved, or owned and not memory-resident, the SSB is eligible for deletion. If the segment is reusable, it is left cached. If it is updatable and modified, it is placed on the write queue. If the segment is not in memory, the swap table entry is released; if in memory, the segment is placed on the loader queue for deallocation. The SSB is then delinked and released. If no more SSBs exist for the associated SGB, SMDSGB is called.

SMDSGB

Deletes a specified SGB. SMDSGB verifies that there are no more SSBs linked onto the SGB and no LUNOs assigned to the associated file, then delinks and releases the SGB. If the SGB is deleted, an >A7 call is placed on the IOUQUE to clean up the file structures.

SMRMVE

Removes a segment from a task. SMRMVE is called when a segment loses its association with a task on the TOL, whether because of a segment manager SVC or task termination. The task-in-memory count for the segment is decremented and, if it goes to zero, the segment is placed on the cache list. SMDSSB is then called to finish processing the removal.

SMFLSH

Writes cached buffer segments to disk and deallocates the memory. SMFLSH processes all segments associated with a specified LUNO (JSB/LDT pair). If they are modified, it places them on the write queue and waits for the write to complete. SMDSSB is called to delete the segment. SMFLSH must be called only by task code.

SMBUFF

Accesses the SSB address of a buffer in a specified task. The caller specifies a JSB, TSB, and buffer address. SMBUFF returns the SSB address for the buffer and the offset of the buffer into the segment.

4.5.7 System Crash Routine.

Whenever an internal operating system error is detected, a branch is made to the system crash routine (NFCRSH), passing a crash code indicating the type of error. The crash routine halts the system and displays the crash code on the front panel of the computer. When the HALT and RUN indicators on the front panel are pressed, the crash routine saves the state of the system at the time of the crash and writes an image of memory to the crash file on disk. This crash file may then be analyzed by systems programmers.

4.6 NUCLEUS FUNCTIONS FOR TASK SCHEDULING AND EXECUTION

The DNOS component that places tasks into execution is the task scheduler (NFSCHD). A task must first be bid and activated before the scheduler can select it for execution. The scheduler selects the highest-priority task ready to execute and causes the central processing unit (CPU) to start executing it. The task then executes for a quantum of time until it voluntarily or involuntarily releases control of the CPU. At this point, the next task in priority order is selected for execution. The execution period may be limited to a value known as a time slice. The scheduler also collects the accounting and performance data related to CPU execution.

The following is a metacode description of the scheduler algorithm:

```
     BEGIN
        IF a task is currently active
        THEN BEGIN
             increment execution time for task;
             IF task is a timesharing task
             THEN BEGIN
             update I/O-bound indicator;
             recompute run-time priority;
             adjust run-time priority for aging;
             END;
             IF task is to remain active
                  THEN requeue task on active queue;
             clear active task;
        END;
        REPEAT
             check for reenter and time-out flags (from DSRs);
             IF DSR task bid is outstanding
                  THEN call task bid routine for task;
             IF a time-delayed task needs reactivation
                  THEN call activate task routine;
             IF any buffered requests need processing
                  THEN call end of buffered request processor;
             IF no task is on active queue
                  THEN idle (wait for next interrupt);
        UNTIL task found to execute;
        set up highest-priority task for execution;
        IF task needs I/O requests unbuffered
             THEN call unbuffering processor;
        place task into execution;
     END
```

4.6.1 Data Structures.

The data structures referenced by the scheduler are JSBs and TSBs. Each JCA includes a queue of TSBs for tasks ready to execute, ordered by execution priority. Each JSB carries the priority of the highest-priority task on its active queue; the queue of JSB

execution. When a task reaches the end of its allotted execution time, its TSB is returned to the JCA active queue if it is to remain active; it is left unqueued if the task is to be suspended. When a task suspends, it may be necessary to change the priority of the highest-priority active task in the JSB and reorder the JSB on the system JSB queue.

4.6.2 Execution Priorities.

Every task has three associated priority values: a run-time priority, an initial priority, and an installed priority. Task run-time priorities range from a high of 0 to a low of 255. The run-time priority is used by the scheduler when selecting tasks for execution. The initial priority is the initial value of the run-time priority and also ranges from 0 to 255. The installed priority is the priority assigned to the task when it is installed in a program file. The calculation of the initial priority is based on the installed priority, the priority of the job in which the task is being bid, and the mode in which the task is being bid (foreground or background). Job priorities range from a high of 0 to a low of 31.

Installed priority 0 is limited to certain system tasks. An installed priority of 0 always maps to an initial priority and a run-time priority of 0. The task's run-time priority does not vary during execution.

Real-time tasks have installed priorities ranging from 1 to 127. The initial priority of a real-time task is always the same as its installed priority. The priority of real-time tasks does not vary during execution. Therefore, the run-time priority is always equal to the initial priority and ranges from 1 to 127.

All other tasks are time-sharing tasks. They have installed priorities of 1, 2, 3 or 4. Installed priority 1 is intended for highly interactive tasks. Installed priority 2 is intended for foreground tasks that are less interactive. Installed priority 3 is intended for tasks that execute exclusively in background. Priority 4 is intended for use by tasks that can run either in foreground or background. Priority 4 is appropriate for almost all user tasks.

The following discussion of initial priority mapping and dynamic priority modification applies only to time-sharing tasks.

Each of the four time-sharing task priority classes (1, 2, 3 or 4) have associated parameters that determine the mapping from installed priority to run-time priority. These parameters can be modified with the Modify Scheduler/ Swap Parameters (MSP) SCI command. The run-time and initial priorities for all background tasks (regardless of their installed priority) are calculated using the scheduling parameters for priority class 3.

The first parameter used in calculating a run-time priority is the Initial Priority Mapping Value. The initial priority for a task is a function of the Initial Priority Mapping Value parameter, the job priority of the job in which the task is being bid, and the Weight of Job Priority parameter. The Weight of Job

Priority specifies the range over which an initial priority can vary based on the job priority. For example, assume that the task being bid has an installed priority of 4 and that the task is being bid in foreground mode. Assume that the Initial Priority Mapping Value parameter for priority class 4 is 190 and that the Weight of Job Priority parameter for class 4 is 32. If the job priority were 0, the initial priority for the task would be 190 - 32 = 158. If the job priority were 31, the initial priority would be 190 + 32 = 222. If the job priority were 7, the initial priority would be 190 - 16 = 174. The mapping from the Initial Priority Mapping Value to the actual initial priority is proportional to the job priority, within the range specified by the Weight of Job Priority parameter.

DNOS has optional dynamic modification of priorities. As a time-sharing task executes, an indicator shows whether the task is I/O-bound or compute-bound. The indicator shows the number of suspensions over a fixed time period and is recomputed at the end of each execution period for a task. This indicator is used to modify the initial priority to create the run-time priority (raising it for I/O-bound tasks and lowering it for compute-bound tasks). The variation of the run-time priority from the initial priority depends on the Dynamic Priority Range parameter for that priority class. A Dynamic Priority Range value of 16 would indicate that the run-time priority could differ from the initial priority by +/-16. A Dynamic Priority Range of 0 would indicate that the run-time priority would never differ from the initial priority.

The default Dynamic Priority Range parameter for all four priority classes is 0. That is, dynamic priority modification is disabled by default. Performance tests have indicated that dynamic priority modification does not improve response time and can cause unacceptable deviations in performance between stations when the computing environment is characterized by homogeneous activity (basically similar tasks executing at most stations). However, dynamic priority modification can improve response time without causing significant performance deviations in heterogeneous computing environments (varied computing activity, possibly occurring at irregular intervals). If a system administrator wishes to try dynamic priority modification, the Dynamic Priority Range parameters should be set to 4,4,0,8 using the MSP command. Dynamic priority modification can always be disabled again by setting the parameters back to 0,0,0,0.

The Aging on Priority parameter is a YES/NO value indicating whether task aging is used for a given priority class. Task aging should only be used for background tasks (priority class 3). If task aging is in effect, the priority of an older task is raised slightly more than the priority of a new task. To raise the priority, the power of 4 that represents the execution time in seconds is used. A task that has executed for 4 seconds is raised 1 priority level, one that executed for 16 seconds is raised 2 levels, etc. Task aging can be disabled by setting the Aging On Priority parameters to NO,NO,NO,NO using the MSP command.

4.6.3 Time Slicing.

Time slicing allows a task to run during a quantum of time and then forces the task to release control of the CPU. This is accomplished by an interface with the clock interrupt processor. The clock interrupt routine counts the number of clock ticks for which a task executes. (A clock·tick is 8.33 MS in the United States, 10 MS in Europe.) When the count reaches a specified number, control returns to the scheduler rather than to the executing task. During sysgen, the user can specify the length of the time slice or can disable time slicing. The length of a time slice can also be changed using the Modify Scheduler/Swap Parameters (MSP) command.

4.6.4 Task Bid.

The process of preparing a task for execution is called bidding a task. This is accomplished by the nucleus routine NFTBID. The process involves building and initializing the necessary data structures, such as the TSB, and activating the task.

4.6.5 Task Activation.

The NFPACT routine activates a task. If the task segments are already in memory, checks are made to see that the task is not being killed and that its job is not terminating; if these conditions are met, the task is put on the active queue. If the segments are not in memory (as is the case following a task bid), the task is put on the waiting-on-memory (WOM) list to be processed by the task loader. (See the section on program management for details.) After the task is loaded into memory, NFPACT is again called to place the task on the active queue.

NFPACT calls the routine NFACTL to place a task on the active queue. The routine NFDACT removes a task from the active queue. The routines NFWOML and NFDWOM place tasks on the WOM list and remove them from the WOM list. The routine NFWOMJ places a JSB on the WOM list.

Figure 4-2 shows the flow through the task scheduler.

4.6.6 Table Area Scheduling.

If a GTA(0) request fails, NFPWOT may be called to place the active task on the Waiting On Table area (WOT) queue. NFPWOT causes the active task's context to be set back as outlined below. NFDACT is called to remove the task from the active list, and NFWOTL places the task on the WOT. NFPWOT then returns through NFSRTN.

When any RTA is executed, the WOT is examined by NFRTA. If a task is on the WOT, NFWAKE is called to restart the task. NFWAKE calls NFDWOT to remove the first waiting task from the WOT and makes it active.

NFPWOT makes certain assumptions about the environment in which the GTA(0) was issued. If the GTA(0) was issued from Map 1 (task) code, NFPWOT expects entry through the GTA(0) error return. The restart context will be set back to the GTA(0) XOP which will be reissued when the task is restarted. If the GTA(0) was issued from Map 0, NFPWOT assumed the failure occurred while processing an SVC. In this case the active task's context is set back to reissue the SVC. This means only modules which process SVC's may call NFPWOT from Map 0. It is also necessary for the SVC processor to restore all system structures to the state they were in before the SVC was issued as it will be reprocessed entirely.

```
                           |
                           V NFSCHD
                     +-----------+
                     | CLEAN UP  |
                     | EXEC TASK |
                     +-----------+
        ------------------>|
        |                  V                     NFTBID
        |              DSR bidding        +---------+
        |                a task?  --yes-->| BID TASK|
        |                  |              +---------+
        |                 no                   |
        |                  |<------------------
        |                  V                     NFPACT
        |              Time delay         +--------+
        |               expired?  --yes-->|ACTIVATE|
        |                  |              |  TASK  |
        |                 no              +--------+
        |                  |                   |
        |                  V----------<--------     NFEOBR
        |              Any requests         +-------------+
        |              to unbuffer? --yes-->| PERFORM     |
        |                  |                | UNBUFFERING|
        |                 no                +-------------+
        |                  |                       |
        |                  V----------<------------
    +------+         Any tasks ready
    | IDLE |<--no---   to execute?
    +------+               |
                          yes
                           |
                           |
                           V
                    +-------------+
                    | SET UP TASK |
                    | TO EXECUTE  |
                    +-------------+
                           |
                           V
```

Figure 4-2 Flow of Control in Task Scheduling

4.7 INTERRUPT PROCESSING

When an interrupt occurs, it is processed by the appropriate interrupt processor. When the interrupt processor is finished, it branches to a return routine (NFTRTN), which returns to either the interrupted code or to the scheduler if the time slice for the task has expired.

4.7.1 Clock Interrupt Processor.

NFCLOK, the clock interrupt processor, gathers performance statistics, keeps track of time, and decides when a time slice occurs. The time and date are kept in the following form: year, day (Julian), hour, minute, second, and tick. Also, a 32-bit tick counter keeps track of time in clock ticks. The time, the date, and the tick counter are updated each clock tick. The tick counter counts clock ticks for 14 months before returning to zero; it is used for timing system functions such as task time delays. (A clock tick is 8.33 ms in the United States, 10 ms in Europe.)

Statistics gathering involves sampling a set of flags. The flags may be set and reset by the operating system at the beginning and end of critical functions. The frequency with which a flag is set determines the percentage of time that the operating system spends within the section of code between the set and reset. A variable contains the number of flags to be sampled; a two-word counter counts the number of times that the flags are sampled. Each flag is a full word and is followed by a two-word counter. The counter is incremented each time the flag is found to be nonzero. The first two flags, representing the CPU and disk utilization, are displayed as a bar graph on the front panel, with CPU utilization in the leftmost eight lights and disk utilization in the rightmost eight lights. This can be changed using the System Configuration Utility. The remainder of the flags are defined to measure other aspects of system performance as shown by the Execute Performance Display (XPD) command.

4.7.2 Internal Interrupt Processor.

An internal interrupt (interrupt level 2) is caused by instruction execution errors (for example, illegal opcode, illegal memory address, or privileged instruction). Internal interrupts are processed by the internal interrupt processor, NFINT2. If the interrupt occurs in task code, the task is killed or placed into end-action code, and control returns to the scheduler. If the interrupt occurs in operating system code, in interrupt processing code, or while scheduling is inhibited, the system crash routine is called.

4.7.3 Power-Up and Power-Down Interrupt Processors.

When a power-down interrupt (interrupt level 1) occurs, the power-down interrupt processor (NFPWDN) idles and waits for the power-up interrupt (interrupt level 0). When the power-up interrupt occurs, the power-up processor (NFPWUP) chains back through contexts saved by interrupt processors (interrupts are not reentered after power up) to find the noninterrupt code that was executing at the time of power down. When the code is found, the map files are set up for that code, the devices are all reinitialized by entering each DSR at its power-up entry point, the microcode is reloaded by calling NFLWCS, and the code is restarted.

4.8 SVC PROCESSING

When a task issues an SVC, the SVC runs with scheduling inhibited until it either completes or suspends the task that issued the SVC. The requesting task is suspended if completion requires a task driven SVC processor.

When an SVC processor terminates, it may reactivate the calling task by branching to NFTRTN. NFTRTN either reactivates the task or, if the time slice has expired, forces rescheduling. SVC processors that suspend the executing task and wish to return to the scheduler do so through the scheduler return routine, NFSRTN. NFSRTN saves the status of the executing task in its TSB and exits to the scheduler.

I/O requests and buffered SVC requests usually require unbuffering of information to the requesting task when the request completes. Unbuffering must occur when the task is in memory. This is accomplished by queuing the buffered request block (BRB), using NFEOBR, to the TSB if the TSB is in memory and to the JSB if the TSB is not in memory. The task may then be activated. Queued BRBs are unbuffered when the task is selected for scheduling.

4.9 TASK TERMINATION

Task execution is terminated when the task issues a termination SVC, another task issues a Kill Task SVC, or the task aborts by executing an illegal or privileged instruction. Task termination is processed by NFTERM. If the task is not terminating normally, NFTERM builds a diagnostic packet and, if the task is active (executable) and has specified end-action (execution after termination), NFTERM restarts the task at the end-action address. The diagnostic packet includes the task program counter, workspace pointer, status, task termination error code, and the time by which the task must finish end action (see the DIA template in the section of data structure pictures).

End action can continue for no more than five seconds, unless the Modify Scheduler/Swap Parameters (MSP) command is used to change the limit

If the task is terminating normally or did not specify end-action, NFTERM deactivates the task (if active), places a task termination entry on the accounting queue, then releases the memory used by the task and system structures that describe that memory by calling NFDTOL and NFDTSK. Finally, if the task was not restarted, an entry is placed on the task termination queue to be processed by the termination processor task, PMTERM. (See the section on program management.)

4.10 SPECIAL COPY ROUTINE

The routines in the module NFCOPY are used to copy blocks of data from one segment to another. There are three main entry points, NFCOPY, NFXCPY, and NFCXFR. NFCXFR is used to copy large blocks of data from one place to another within the current map file. It can be used in either map file 0 or map file 1. NFCOPY and NFXCPY are used to copy data from one segment to another where neither of the blocks need be mapped. NFXCPY must be called from map file 0 and NFCOPY must be called from map file 1 through the NFMAP0 interface. NFCOPY calls NFXCPY which then calls NFCMAP to set up a special map file which is used for a call to NFCXFR. The routine NFCMAP can be called to set up map files for special purposes by other routines which run in map file 0 and are located in the system root.

SECTION 5

IPL AND SYSTEM LOADERS

5.1 IPL SEQUENCE

The DNOS initial program load (IPL) process consists of several logical steps:

1. A read-only memory (ROM) loader on the CPU loads the track 1 loader (a simple bootstrap program).

2. The track 1 loader loads the system loader.

3. The system loader loads the operating system and any memory-resident tasks from the user's application program file.

ROMs are discussed in other documents about the 990 computer. See, for example, the Universal ROM Loader User's Manual.

After being loaded, the track 1 loader relocates itself to the last 8K bytes of the first 64K bytes of memory and then reads the disk volume information from track 0, sector 0. From this information, the track 1 loader determines whether it is to load a diagnostic (stand-alone) program, a secondary loader, or an operating system. The file to be loaded may be either an image file or an object (compressed or noncompressed) file. After determining what is to be loaded, the track 1 loader loads the program into a portion of the first 64K bytes of memory, starting at address >A0. Note that this loader cannot load any program larger than 54K bytes.

The system loader loads DNOS from the kernel program file, using the steps shown in Table 5-1.

After the system is loaded, the loader passes control to the power-up interrupt handler of the loaded operating system.

The following paragraphs describe in more detail the operation and logic of the DNOS system loader, as well as the data structures used by the loader.

5.2 SYSTEM LOADER OVERVIEW

The system loader resides on disk in an image file called DSC.S$IPL and is loaded into memory by the track 1 loader. It is linked as if it were a system task; that is, it expects to be mapped in with the operating system root and a JCA while executing. This allows the loader to call subroutines in the root after the root has been loaded into memory. The loader executes with interrupts masked to level 2, inhibiting interrupts from devices.

Once loaded into memory, the loader enables mapping, creating for itself a two-segment map file. The first segment contains the loader code, which is located in the first 8K bytes of physical memory. The second segment maps in the 8K bytes of physical memory immediately following the loader code.

The first section of code (located in module SLIPL) initializes physical memory to reset any correctable memory errors and to determine the actual size of physical memory. This procedure involves writing to each word mapped into the second segment, changing the map file to map in the next 8K bytes, and writing into each word in that segment. This process is repeated until the loader tries to write to memory that does not exist.

Having found the end of physical memory, the loader maps in the last 16K bytes as its second map segment and relocates itself to that segment resetting its map file such that the first map segment maps in the memory starting at physical address 0 and logical address 0, and the second map segment maps in the memory containing the loader code, starting at logical address >C000. From this point on, as the loader finishes a particular phase of the load process, it displays the phase on the front panel lights, starting at the left. Table 5-1 lists the different phases and indicates the significance of each.

Table 5-1 System Loader Phases

Phase	Description
1	Successful relocation of the loader
2	Successful open of kernel program file
3	Successful load of root, verification of system version, and load of writable control store (WCS)
4	Successful load of special table areas
5	Successful initialization of system overlay table and crash file
6	Successful load of JCA segments
7	Successful load of DSRs and scheduler
8	Successful load of memory resident system tasks
9	Successful load of memory-resident user tasks

Next, the loader initializes its load device (disk drive) for I/O. It then determines whether the machine being loaded is a 990/12.

The system root, consisting of a procedure and a task segment from the kernel program file, is then loaded into memory, starting at location 0. The loader creates a new three-segment map file, mapping in the root as the first segment, the following physical memory (up to address >C000) as the second segment, and the loader code as the third segment. As soon as the root is loaded, the loader verifies that the loader file (.S$IPL), the kernel program file, and the utilities program file (.S$UTIL), are all of the same version. Then the loader checks the volume information from the disk being loaded to see if a writable control store (WCS) file is specified. If so, it then loads the WCS from the file.

Next, the loader loads or creates the memory-based segments of the operating system. The loader traverses the memory-based SSB list located in the STA. Each SSB represents a file management or segment management table area and indicates whether to load a segment from the kernel program file or to build a segment in memory (a nonzero SSBADR value indicates that the overlay is to be loaded from the kernel program file). After loading or creating a segment, the loader initializes that segment's overhead words.

The loader then performs the following:

* Determines which of the disk drives defined is the disk from which the system was initially loaded and marks it as the system disk

* Installs the system disk

* Initializes the system overlay table

* Builds the file structures for the swap file and the crash file

After all of the special table areas are in memory, the loader scans the JSB list in the system table area. Each JSB points to an SSB for a JCA that needs to be loaded from the kernel program file. JCA segments may also require name segments; if so, the loader creates the segments. Table management overhead words are initialized in both JCA and name segments.

The next phase consists of loading the DSRs, the scheduler, and the SVC processor segments. The map files of these various segments, which are in an array for the scheduler and in the physical device tables (PDTs) for the DSRs, contain the installed IDs of the overlays on the kernel program file. The loader scans the map files, loading any segments indicated.

The loader then reads the memory-resident system task bit maps from the kernel program file and the utilities program file, loading each task indicated. Any associated procedures are also loaded. SSBs are created and initialized for all segments loaded in this phase. If a user application program file was specified during sysgen, the loader reads the bit map for that file and loads all memory-resident tasks, procedures, and segments.

The next step in the load process is installing all on-line disk volumes. Installing a volume includes initializing PDT information, creating an FDB for VCATALOG for that disk, and initializing the disk manager data structures.

The final phase of the loader execution allocates the buffer table area (BTA), loads the I/O utility task, and initializes the system anchor for BTA. BTA is located in user memory, immediately following the memory-resident portion of the operating system and all memory-resident tasks. The I/O utility task is then loaded, and the system anchor is initialized for the file memory list. The memory containing the loader is part of user memory. After the initialization is performed, the loader transfers control to the power-up interrupt processor of the operating system.

5.3 SYSTEM LOADER DATA STRUCTURES

Since the data structures created by the system loader are also used by other parts of the operating system, the data structures themselves are not described in detail. The loader's use of these structures and the reasons for their existence are described in the remainder of this section. The descriptions assume that the load medium is a disk. In the loader modules, device-dependent code is localized to as few modules as possible. As a result, the loader is easily configurable as a download program that uses a communication port as its load device.

The system loader uses the following data structures on the disk:

* Volume information (track 0, sector 0)

* Volume directory (VCATALOG)

* Kernel program file, named during sysgen

* Utilities program file, .S$UTIL (or a name chosen by the user)

* Shared program file, .S$SHARED

* Application program file

* Writable control store (WCS) file

* Partial bit maps (while installing the disk)

All except the volume information and the WCS file are standard structures, as described in the section on data structure pictures.

In addition, the system loader uses modules SLDATA and SLDISK for internal working storage. These storage areas are part of the system loader object itself, and are available to the system loader for the duration of its execution.

5.3.1 Disk Volume Information.

The volume information contains the following data used by the system loader:

* Starting allocatable disk unit (ADU) of VCATALOG, the volume directory

* Names of the following files:

  - kernel program file

   - utilities program file
   - WCS file (If the Performance Package is present)

- Total number of ADUs on the disk
- Starting sector of the partial bit maps
- Volume name
- Number of sectors per ADU on the disk

5.3.2 WCS File.

The WCS file is an image file whose content is of the following form:

- Word 1 - number of bytes of overhead
- Word 2 - microcode word size
- One or more repetitions of
   - Word 3 - microcode starting address
   - Word 4 - number of microcode words
   - Microcode

5.3.3 Kernel Program File.

Although the kernel program file is standard in format, its contents are slightly unusual. The kernel program file is created by the Assemble and Link Generated System (ALGS) portion of sysgen and contains all of the system segments that are configurable during sysgen. The file contains the following:

- System root (two procedure segments)
- System JCA (overlay)
- First segment management table area (overlay)
- All DSRs included during sysgen (overlays)
- Configurable system tasks (starting with task ID 2) and their overlays
- JCAs for sysgen-defined jobs (overlays)

5.3.4 System Loader Internal Working Storage.

The modules SLDATA and SLDISK contain the following data items local to the system loader.

* SLDATA

  - System loader MAP files

  - Linkage to system file structures

  - Memory management and allocation information

* SLDISK

  - Disk initialization routine (SLINIT) workspace

  - Disk I/O routine (SLDIO) workspace

5.4 FLOW OF CONTROL THROUGH THE SYSTEM LOADER

SLIPL is the main routine of the system loader. It includes the loader relocation code and calls to subroutines that perform all of the actual loading. Figure 5-1 shows the calling relationships between the different loader modules.

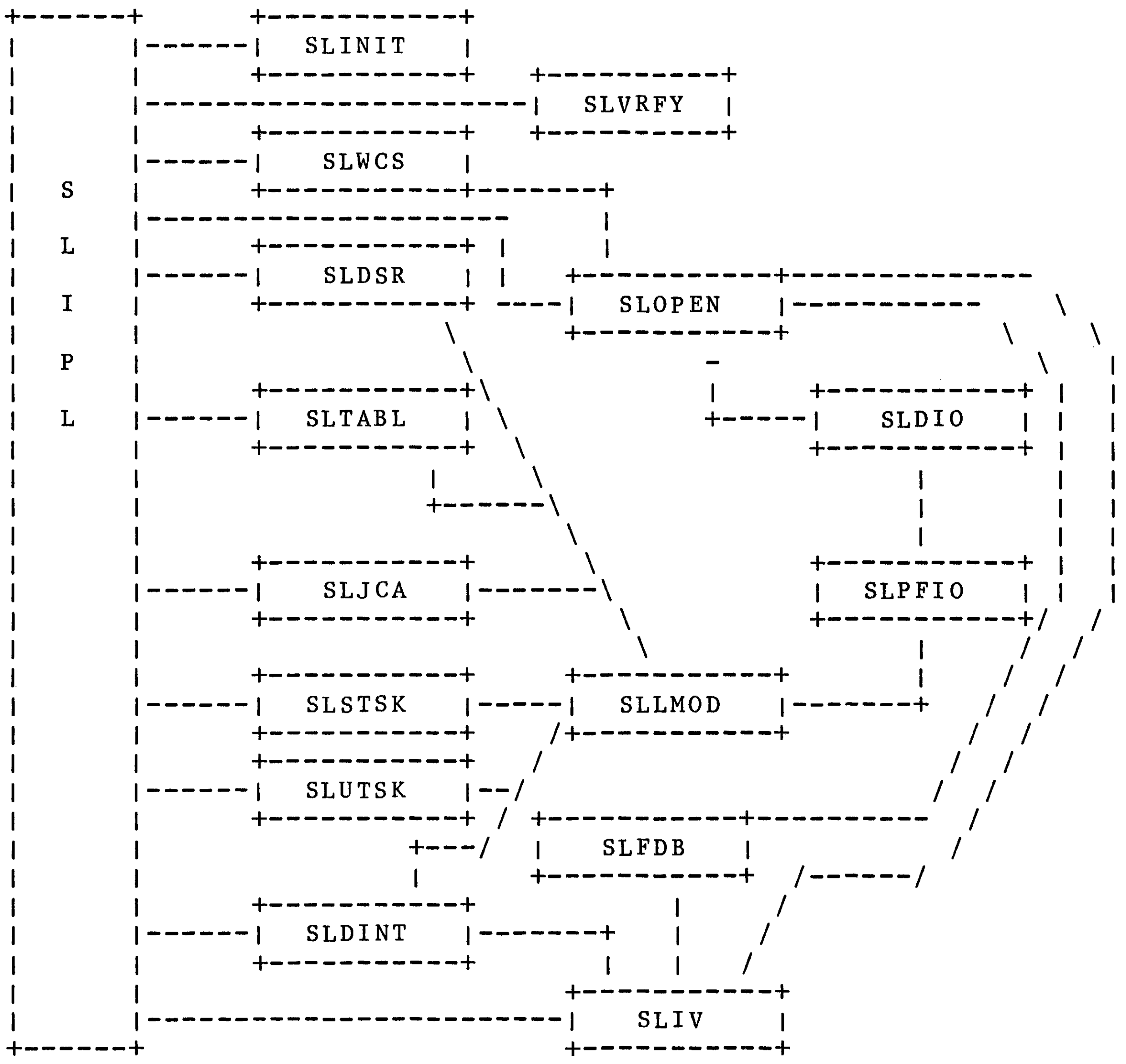


Figure 5-1 System Loader Subroutine Calls

5.4.1 Relocating the Loader.

As described in the overview of the system loader, the first activity of SLIPL is to determine the size of physical memory. This is accomplished by using a second map file segment (the first segment maps in the loader code). Initially, the loader map file maps memory as shown:

```
 1st segment  2nd segment
+-----------+-----------+----------------/
|           |           |                \
|   loader  | 8K bytes  |                /
|           |           |                \
+-----------+-----------+----------------/
0
```

The memory initialization code then writes to every word in the second map segment, comparing the contents of each word after the write to verify that the contents are the same. If the comparison fails, the loader assumes that it is at the end of physical memory.

After 8K bytes have been checked, the loader resets its map file as shown:

```
 1st segment             2nd segment
+-----------+-----------+-----------+-----/
|           |           |           |     \
|   loader  | 8K bytes  | 8K bytes  |     /
|           |           |           |     \
+-----------+-----------+-----------+-----/
0
```

This process is repeated until the end of physical memory is found.

NOTE

> If the computer being loaded contains the maximum amount of memory allowed, or if the search for the end of memory causes the loader to write to the TILINE peripheral control space (TPCS), the write/compare test will fail on the first write to the TPCS; thus, no accidental TILINE commands can be issued. (TILINE is a registered trademark of Texas Instruments Incorporated.)

After finding the end of memory, SLIPL relocates the loader to the upper 16K bytes of physical memory, mapping memory as shown:

```
 1st segment                          2nd segment
+------------+---------//----------+------------+
| old        |         \\          |            |
|loader      |         //          |   loader   |
| code       |         \\          |            |
+------------+---------//----------+------------+
0           >C000                          end of memory
```

After the relocation, SLIPL calls SLINIT to initialize the load device for I/O.

SLIPL also determines the CPU type and saves it as CPUID in NFDATA.

5.4.2 Load Device Initialization.

SLINIT has two entry points, SLINIT and SLIVSU. SLINIT performs some device initialization, dependent on values found in the loader ROM workspace (location >80 through >9E), and is called by SLIPL. SLIVSU is an entry point used by the disk installation routine, SLIV, to gather the information about a disk drive necessary to install the volume. The device initialization logic consists of performing a Store Registers command to the disk drive and then reading the volume information (track 0, sector 0). From this information, SLINIT initializes the workspace used by the disk I/O routine (SLDIO), saves the important file names, and saves the ADU address of VCATALOG. Since the device information is saved in common segments, it is accessible by the other loader routines.

5.4.3 Opening a File for I/O.

Before loading the system root, SLIPL calls SLOPEN to open the kernel program file for I/O. SLOPEN is an important routine in the loader; it accepts as input a file name, which is assumed to be cataloged in the volume directory VCATALOG. It then calculates the hash value of the file name and searches VCATALOG for the File Descriptor Record (FDR) for that file. When the file is found, SLOPEN reads the FDR into the loader's internal buffer (located after the last module in the loader) and then builds a file control block (FCB) and file descriptor block (FDB) for the file. The FCB and FDB are built in the file manager table (FMT) if the FMT has been loaded, otherwise, they are built in a temporary area in one of the loader common segments. The FCB information is used by the program file I/O routine, SLPFIO, to read and write to the file on disk.

NOTE

> The loader is designed so that it can perform I/O to only one file at a time; in other words, only one file can be open at a time.

5.4.4 Loading the System Root.

After the kernel program file is open, SLIPL loads the system root. It calls the module load routine, SLLMOD, to load procedure 1 and procedure 2 from the kernel program file. These two modules are loaded in adjacent memory, starting at location 0, and combine to form the system root segment. After the root is loaded, SLIPL resets its map file to be a three-segment map file. It maps the root as the first segment, the physical memory immediately following the root as the second segment, an the loader code as the third segment, as shown:

```
1st segment  2nd                         3rd segment
+---------+------+-----//----------+------------+
| system  |      |     \\          |            |
|  root   |      |     //          |   loader   |
|         |      |     \\          |            |
+---------+------+-----//----------+------------+
0         JCASTR  >C000                 end of memory
```

After loading the root, SLIPL calls SLVRFY to verify that the versions of the kernel program file, the utility program, and S$IPL match.

5.4.5 Loading a Module.

The loader calls SLLMOD to load a segment (task, procedure, or overlay) from the currently open program file. The module is always loaded into memory, starting at the next available beet address. (A beet address is an address evenly divisible by 32.) Memory is allocated linearly from physical location 0 to the end of memory. SLLMOD is used for three purposes:

* Loading a kernel segment (a segment that is not a system or user task, such as a JCA or a DSR)

* Loading a task or procedure segment

* Reading the program file directory index (PFI) for a segment

When loading a kernel segment, SLLMOD does not create any system overhead (such as an SSB or OVB for the segment). It does, however, make an entry in an internal table to indicate which kernel segments have already been loaded. Thus, if a segment is requested more than once (as is the case for the system JCA) it will be loaded only once. This internal table has the following form:

```
  +-----------------------------------------------+
0 | type  |  ID    |    load beet    |  seg. length |
  +-----------------------------------------------+
1 |                |                 |              |
. +-----------------------------------------------+
. //               //                //             //
. +-----------------------------------------------+
n |                |                 |              |
  +-----------------------------------------------+
```

Each table entry is three words long and contains four fields as follows

* The first byte of the first word is the segment type (0=task, 4=procedure, 8=overlay) on the program file. Note that a segment installed as a procedure or a task on the kernel is not necessarily loaded into memory as a procedure or task. The system root is an example of this.

* The second byte of the first word is the installed ID on the program file.

* The second word is the beet address where the segment was loaded.

* The third word is the byte length of the segment.

When a kernel segment is requested, SLLMOD first searches the table to determine if the segment is already loaded; if so, SLLMOD immediately returns the load beet and segment length to the caller.

If the segment requested is a task or procedure segment, SLLMOD loads the segment and builds system overhead for it (SSB and OVB). Before trying to load the segment from the program file, SLLMOD calls a routine in the system root, SMFSID, to search the SSB group for the SSB of the currently open program file. If the SSB is found, the segment is already in memory and need not be reloaded; otherwise, the segment must be loaded.

5.4.6 Initializing the Crash File.

After the system root is loaded, SLIPL calls SLCRSH to initialize the crash file information in the system root. This information is kept in the CSEG NFDATA and consists of the TILINE address; the head, cylinder, and sector addresses of the crash file; and the size of the crash file (in beets). SLCRSH obtains the information by opening the file and extracting information from the FDR for the file.

5.4.7 CPU Type Dependent Initialization.

After the load device is initialized, SLIPL determines the CPU type. This is done by examining a CRU location. If the CPU is a 990/10 or a 990/10A, no special initialization is done. If the CPU is a 990/12, SLWCS is called to load the WCS file if one is specified in the volume information. If the CPU is an S300, the clock handler is initialized for a 50hz clock.

5.4.8 Loading the Special Table Areas.

The special table areas for segment management, file management, and system common are represented by SSBs in the memory-based segment group, located in the STA in the root. These SSBs, built during sysgen in the $BLOCK module in the D$SOURCE file, can be initialized with either of the following formats:

- * The beet address field of the SSB contains an overlay ID
- * The beet address is 0 and the length field contains the length of the segment to be created.

Only the first segment management table area SSB and the system common SSB are of the first format; none of the others represent actual program file segments.

If the table area is a segment in the program file, it is constructed during sysgen to include only the defined data area, thus occupying less disk space than if free area was also allocated. The SSB for the table area contains the correct length in the SSBLEN field. SLLMOD allocates the difference between the SSBLEN value and the segment installed length as free table area. When the system loader loads one of these segments, it adds the size of the free area to the memory already allocated for the segment; the result is a segment in memory that includes all of the free area.

If the segment has no program file image (it is completely empty and so sysgen built only an SSB for it), SLTABL allocates the amount of memory indicated in the length field of the SSB; SLTABL then initializes the table area management overhead words in the segment to indicate that it is completely empty.

5.4.9 Loading the JCAs.

The SSBs that represent JCA segments are also in the memory-based segment group but are not located in the STA in the root. They are located in the first segment management special table area, which is built during sysgen and loaded in the preceding phase of the system load. To load the JCAs into memory, SLIPL calls the routine SLJCA. This routine scans the JSB list, maps in the segment management special table area and then uses the SSBADR field to indicate which segment is to be loaded. SLJCA never creates a JCA segment, since they are all built during sysgen and have a segment in the kernel program file.

NOTE

> Normally, JCA segments are considered swappable (except for the system JCA).

As SLJCA loads each JCA segment, it inspects the job information table (JIT) in the JCA to see if any name segment must be created for the job. This is indicated by a nonzero value in the SSB address field for the segment. If the value is nonzero, it is used as the size of the area that must be created. SLJCA creates an empty segment and initializes it as a name segment. (For details, see the description of name management in the section on the I/O subsystem.)

5.4.10 Loading the DSRs.

The next phase of the load process is the loading of the DSRs, the scheduler, and the SVC processor segments. The routine SLDSR loads these. SLDSR first loads the scheduler and SVC processor segments, then the DSRs. It determines which segments to load by inspecting the map files for the scheduler and DSRs.

The scheduler/SVC map files are in an array located in the STA in the root. The array begins with the scheduler map file, and MAPSHD in the NFPTR common segment in the root points to the array. Each entry in the array is a six-word map file, initialized during sysgen as follows:

1. Limit 1 is set to the length of the root.

2. Bias 1 is set to 0.

3. Limit 2 is set to >4000 (one's complement of >C000).

4. Bias 2 is the overlay ID of the system JCA.

5. Limit 3 is set to the negative value -1. (This is a signal used by IPL to determine whether or not the DSR map file has been initialized.)

6. Bias 3 is the overlay ID of the scheduler or SVC segment to be loaded.

SLDSR inspects each map file, loading the segments indicated by the bias 2 and bias 3 fields and initializing each map file with the correct bias and limit values.

After the map file array has been processed, SLDSR scans the PDT list, loading the segments indicated by the map file in each PDT. The PDT map files are initialized in the same way as the SVC/scheduler map files, with the value in bias 3 being the overlay ID of the DSR for the device.

5.4.11 Loading Memory-Resident Tasks.

After all of the JCAs are in memory, SLIPL is ready to load all of the memory-resident tasks for the system and for user jobs. SLIPL first calls SLSTSK to load all of the tasks defined in the system job. SLSTSK calls SLMRES to load all memory-resident tasks on the kernel program file. SLSTSK then opens the utility program file and calls SLMRES to load all memory-resident tasks in that program file. SLIPL calls SLUTSK to load user-defined tasks from the user's application program file. SLUTSK operates in the same manner as SLSTSK.

5.4.12 Disk System Initialization.

SLIPL calls SLDINT to perform some system disk initialization. SLDINT performs the following functions:

1. Searches the PDT list for the disk PDT, which represents the disk from which the system was loaded. This PDT is then marked to be that of the system disk by setting the system disk flag and setting the pointer SYSPDT to point to the PDT.

2. Opens the system swap file by calling SLOPEN.

3. Installs the system disk volume by calling SLIV.

4. Initializes the system overlay table used by the system overlay loader.

5.4.13 Installing Disk Volumes.

The next phase of the system loader is the installation of all disk volumes that are on-line during IPL. To do this, SLIPL calls SLIV, which scans the PDT list in the STA, searching for a disk.

SECTION 6

SVC REQUEST PROCESSING

6.1 OVERVIEW OF SVC PROCESSING

In the 990 hardware architecture, 16 levels of extended operations (XOPs) are defined. Level 15 is reserved for use as an interface between user software and operating system services. This interface is named the Supervisor Call (SVC) interface.

When an SVC is issued, the 990 computer hardware transfers control to a software routine, which begins decoding and processing the SVC. The activity of the decoding routine varies, depending on the particular SVC request. Some SVCs are processed quickly, with little information passed from requester to processor. Other SVCs require extensive effort and time or require much information transfer between requester and processor. To allow optimum use of the 990 resources, an SVC that requires much time to process is copied into a block of system table area (STA) along with information identifying the requester; then the requester task is suspended and its memory is relinquished to other tasks.

The amount of effort involved plus several other factors determine the method used by an SVC processor. The SVC request is copied (buffered) into registers if the processor meets the following conditions:

* It is a memory-resident processor

* It completes processing of the SVC in a short period of time

* It processes an SVC that may be issued by any task

* It processes an SVC that cannot be an initiated event (using SVC >41)

* It returns all results directly to the requester task space

Otherwise, the SVC request is buffered into the STA.

While a task is having an SVC decoded, that task cannot lose its time slice or be preempted by the scheduler. When the SVC issued is one that finishes quickly, the request is decoded and is processed, and control returns to the requester task before the scheduler can schedule another task for execution. Essentially, the sequence of events is as follows:

1. Requester task issues the SVC by using XOP @block,15 (or equivalent)

2. Decoding routine is entered from the XOP interface

3. Decoder determines that this is a request which finishes quickly

4. Decoder copies some or all of request block into processor routine registers

5. Decoder transfers control to processor

6. Processor performs requested service and returns information to requester task

7. Processor returns control to requester task

If the SVC request issued is not a fast request, the SVC decoder copies the request block into a buffer in STA and then follows one of two possible paths. For requests that require much time and effort, usually the request is queued to a processor task and the requester task is suspended until the request completes. Processors that are disk-resident tasks follow this path. Such processors are either seldom used or very large in size.

Certain special processors, such as those for I/O and job management use an alternate path for processing buffered requests. Some preprocessing is required before control goes to a processing task. When following this path, the decoder copies the request block into a buffer in STA or JCA and transfers control to the preprocessor. The preprocessor examines the buffered request and performs whatever processing it can. For some subopcodes, all processing is completed in the preprocessor. In these cases, control is returned to the requester task. In other cases, the preprocessor queues the request to the processor task and suspends the requester task.

When the requester task is suspended while the SVC is being processed, the requester task may be removed from memory to make room for another task. When the SVC request is finished, the buffered request must be returned to the requester task; then the requester task can again be scheduled for execution. To allow this, the SVC processor queues the finished buffered request block to the requester task's TSB (or to the task's JSB if the TSB is not in memory). When ready for a new task, the scheduler examines these blocks, ensures that the task is in memory, calls a routine to return information to the task, and schedules the task for execution.

The decoding routine examines the SVC request not only to determine whether processing will be fast or slow, but also to verify several other characteristics. Some SVC requests must be aligned on a word boundary in order to execute properly. This is the first characteristic the decoder checks for. If the request block is not aligned but should be, an error code of >F1 is returned in the return code field of the request block, and the requesting task resumes control.

Another characteristic to be checked is the privilege level of the request. Some SVC requests can only be issued by operating system tasks. If this requirement is not met, an error code of >F2 is returned in the return code field of the request block, and the requesting task resumes control. Some SVC requests require that the requesting task be installed as software privileged. If this requirement is not met, the task receives an error code of >F3, and the requesting task resumes control.

Since some of the SVC requests (and their processors) are configurable when a DNOS system is generated, it is possible for a task to issue an SVC that is not supported on a particular DNOS system. When this occurs, an error code of >F0 is returned to the request block, and control returns to the requesting task. This error code is also returned when a request specifies an SVC code that is not defined in the DNOS set.

Some DNOS users extend the capabilities of the operating system by adding their own SVC codes and processors during sysgen. (Such user-defined SVCs have operation codes >80 or greater.) The same checks are performed on user-defined SVCs as on the DNOS SVCs, and the same set of error codes is used for these checks.

6.2 MODULES USED FOR REQUEST PROCESSING

Most of the routines for processing SVC requests are written in 990 assembly language; several are written in Pascal. The routines are found in modules either in the subsystems that they directly support, or in the DSC.REQPROC directory. Modules in REQPROC support the decoding, buffering, and unbuffering of requests and also process some of the requests that do not belong in any other DNOS subsystem. Table 6-1 lists and describes some of the request processor modules found in the REQPROC directory.

Table 6-1 Major Request Processor Routines

Name	Description
RPBUF	Routine that copies request blocks to buffers in STA
RPCONV	Processors for SVC 0A,0B,0C,0D (data conversion)
RPDQUE	Routine that dequeues and unbuffers SVC requests to requester tasks
RPGSVC	Processors for SVC 02,03,06,07,09,0E,11,2E,2F,35, 3B,3E (miscellaneous general-support SVCs)
RPIDSC	Processor for SVC 38 (Initialize New Disk Volume)
RPINV	Main driver for the initialize new task volume
RPINV1	Routines used to support the initialize volume process
RPINV2	Same as RPINV1
RPINV3	Routines used to initialize disc process
RPINV4	Utility routines for the initialize new volume process
RPIOR	Utility routines for the IV, UV, and INV SVC handlers
RPIV	Handles the main portion of installation of a disc volume
RPPRCK	Routine that checks for memory protection violations
RPPEVT	Processor for SVC >4F (Post Event)
RPPSVC	Processors for SVC 04,10,1B,24,2B,2C,33 (miscellaneous) (program-support SVCs)
RPRCDA	Data base for SVC 4C (Return Code Processor)
RPRCP	Processor for SVC 4C (Return Code Processor)
RPRETR	Processor for SVC 3F (Retrieve System Data)
RPROOT	Decoder for SVC requests
RPSDAT	Module that includes the system static buffer and a table (RPSTAB) built during sysgen, showing characteristics and processors for DNOS SVCs
RPSGCK	Routine that checks for mapping violations
RPUDAT	Includes the table RPUTAB built during sysgen, showing characteristics and processors for user-defined SVCs
RPUTIL	Utility routines and data areas for the IV, UV, and INV SVC handlers
RPVOL	Processors for SVC 20,34 (Install Disk and Unload Disk)
RPWAIT	Processor for SVC 42 (Wait for Event)
RPWTIO	Processors for SVC 01,36 (Wait for I/O)

Other modules that process SVC requests are found in the subsystems for I/O, name management, job management, program management, and segment management. Short descriptions of the routines that process SVCs can be found in the relevant subsystem descriptions.

6.3 MAPPING STRUCTURE

Due to the large number of SVC processors, one map file segment cannot contain all of them. Therefore, two arrangements of map file 0 are set up during sysgen. One arrangement has these three segments mapped in: system root, requester JCA, scheduler/first SVC segment. The other arrangement has these three segments mapped in: system root, requester JCA, second set of SVC processors. A flag in the RPSTAB entry shows which of the map arrangements is needed for processing a particular SVC. Before passing control to the processor, the decoding routine makes sure that the correct map file is being used. When the processor terminates, the return routines ensure that the map file with the scheduler segment is restored.

6.4 DATA STRUCTURES USED FOR SVC PROCESSING

The primary structure used by the SVC decoding routine is the SVC definition table built during sysgen. This table, RPSTAB, is created to define completely all DNOS SVCs included in the current system configuration. Users who supply any of their own SVCs must build a similar table, RPUTAB, to describe those SVCs. The RPSTAB table is located in a module named RPSDAT; the user defined table is placed into a file named .S$SGU$.USERSVC.RPUDAT.

Each DNOS-supported SVC code has a two-word description field in RPSTAB. For codes that are undefined in a particular configuration, both words are zero. Figure 6-1 shows the two-word description format.

```
BYTE 0 - FLAG BYTE
        BIT 0 - 0= Do not check alignment; 1=check alignment
            1 - 0= Use registers to buffer; 1=use table area
            2 - 0= Use first SVC segment of processors;
                1= Use second SVC segment of processors
          3,4 - Reserved
          5-7 - Length to buffer, if going to registers;
                otherwise 0
BYTE 1 - LENGTH BYTE
         >00 if buffering in table area
         Length of whole call block if buffering in registers
BYTES 2,3 - ADDRESS WORD
         Address of request definition block (RDB) if
         buffering in STA
         Address of processor if buffering in registers
```

Figure 6-1 SVC Entry Form in RPSTAB

SVC processors that execute quickly and require little information from the SVC call block have the required information buffered in registers. Upon entry to the SVC processor, the following registers are set:

- R0 - bytes 0,1 of call block (or zero if unused)
- R1 - bytes 2,3 of call block (or zero if unused)
- R2 - bytes 4,5 of call block (or zero if unused)
- R3 - requester call block address
- R4 - requester TSB address
- R5 - requester map file pointer in TSB

When using a buffer in STA, a structure called the request definition block (RDB) is used to tell how much and which fields to buffer. The RDB is defined in the module with the memory-resident processor or preprocessor, if one is used. For SVCs processed by tasks with no preprocessors or for SVCs that are configurable options of DNOS, the RDB is defined in the RPSDAT module. The RDB is labeled RDBSxx for system SVC opcode xx. A template for the RDB is shown in the section on data structure pictures. Figure 6-2 shows examples of RDBs.

For many of the requests buffered according to an RDB, information must be returned from the processed buffered request to the requesting task. The structure used to govern this transfer is the return information block (RIB) built for the SVC. A RIB is needed if information in addition to the return code must be passed back to the requester task. The RIB for system opcode xx is RIBSxx and is shown in detail in the section on data structure pictures. Figure 6-2 shows an example of an RIB.

```
RDBS14 EQU  $                   LOAD OVERLAY RDB
       DATA >0800               USE DYNAMIC BUFFER IN STA
       DATA OVYQUE              OVERLAY QUEUE SERVER HEAD
       DATA 0                   NO RIB NEEDED
       DATA >0007               MAX BUFFER SIZE
       BYTE >07                 BASIC BLOCK LENGTH
       BYTE 0                   ACCOUNTING FACTOR
       DATA 0                   RESERVED

RDBS48 EQU $                    JOB MANAGER RDB
       DATA >1800               PREPROCESSOR, DYNAMIC BUFFER
       DATA JMPREP              ADDRESS OF PREPROCESSOR
       DATA RIBS48              RIB ADDRESS
       DATA >0010               MAX BUFFER NEEDED IS 16 BYTES
       BYTE >10                 BUFFER 16 BYTES
       BYTE 0                   ACCOUNTING FACTOR
       DATA 0                   RESERVED

RIBS48 EQU  $
       DATA 0                   NO POST PROCESSOR NEEDED
       BYTE 0                   START UNBUFFERING AT BYTE 0
       BYTE >10                 UNBUFFER 16 BYTES
       DATA 0                   END OF RIB
```

Figure 6-2 Examples of RDB and RIB Structures

The job management SVC is one example of an SVC that must be rebuffered for certain sub-opcodes. The flags defined in the RDB for expansion govern that rebuffering. This technique is used because request blocks for sub-opcodes within the SVC opcode vary in size. The preprocessor of the SVC must make a call to RPBUF with a revised RDB to rebuffer special cases.

SVC processing uses several data structures in addition to the RDB, RIB, and RPSDAT. Among these are the queue headers for the queue server SVC processing tasks. The queue headers are described in the section on nucleus functions. The SVC decoder uses the queue header pointer in the RDB to queue a buffered request to a queue server.

SVC processing uses TSBs of the requesting tasks to access map file information and to return completed requests. It uses JSBs to return completed requests if the TSBs are not available. Other structures are used by particular SVC processors but not by the decoder or buffering routines.

6.5 DETAILS OF SVC PROCESSING

SVC processing begins in the routine RPROOT. This routine accesses tables to locate the appropriate processor and to determine buffering details. The routine RPBUF is used to copy (buffer) the SVC request into a temporary work area. The routine RPDQUE is used to return the finished request to the task issuing the SVC. A set of miscellaneous routines is used throughout processing.

6.5.1 Decoding Routine (RPROOT).

When an SVC is executed, the hardware transfers control via the interrupt processing routines to the SVC decoding routine RPROOT. RPROOT first checks for a special SVC (XOP 15,15) used by the SCI Debugger. If this special call was issued, a flag is set in the requesting task's TSB.

A check is then made for the Initiate Event SVC. If that SVC is specified, it is now processed in RPROOT. The SVC being initiated is checked to ensure that it is a legal opcode and can be initiated. (In the current version of DNOS, only I/O and semaphore operations can be initiated.) If no errors occur, the initiated SVC is processed like any other request.

At this point I/O and Segment Manager SVCs are checked for alignment and then routed directly to their preprocessors. This is done to speed up the processing of those SVCs.

RPROOT then examines the RPSTAB entry for the requested SVC. The first check verifies that the opcode is defined in this configuration. If there is no RPSTAB entry and no RPUTAB entry, error code >F0 is returned, SVC processing terminates, and control returns to the requester task.

If the SVC is defined, the next check is for alignment. If the RPSTAB or RPUTAB entry shows that the request must be aligned on a word boundary, the address of the request is checked. If it is not legal, error code >F1 is returned, and SVC processing terminates.

The RPSTAB entry for the requested operation is checked to see whether buffering occurs in registers or in STA. If the request is to be buffered in registers, RPROOT performs the following:

1. Checks the call block for mapping and protection violations

2. Transfers the required amount of information from the requester call block to registers R0, R1, and R2

3. Ensures that the correct map file is in use

4. Transfers control to the processor. When the processor completes its work, it transfers control back to the requester task

If the RPSTAB entry for the SVC shows that the SVC is to be buffered into STA, RPROOT performs the following:

1. Accesses the RDB

2. Checks the call block for mapping and protection violations

3. Calls the buffering routine RPBUF to transfer information from the requester call block to STA according to the RDB, creating a BRB (Buffered Request Block)

4. Checks whether the request is to be queued to a queue header for a task or sent on to a processor in memory

   a. If the request is to be queued, RPROOT queues the buffered block to the queue header and suspends the requester task

   b. If the request is to be sent to a processor, RPROOT transfers control to the processor, which either returns to the requester or queues the buffered request to a task

After RPROOT transfers control to an SVC processor, that processor may return control to the scheduler by branching to NFSRTN or NFTRTN. It may also return to RPROOT in case an error occurs in the processing logic. The return points are as follows:

* RPRTNE - an error completion. RPROOT must check whether this was an initiated event and return only the error byte to the requester task.

* RPRTNF - a task error in the requester task. RPROOT must check whether this was an initiated event and terminate the requester task with the task error passed from the SVC processor. If the SVC processor itself encounters a logic error, RPROOT terminates the requester task with task error >04 to show an SVC processor error.

6.5.2 SVC Buffering Routine (RPBUF).

RPBUF is a general request buffering routine called by RPROOT and by several SVC preprocessors that have received a partially buffered block from RPROOT. RPBUF uses the RDB provided by the caller to determine how to buffer the information.

RPBUF first checks the RDB flags to see if this buffering is to use the single system static buffer (provided as part of the RPSTAB module) or a dynamic buffer. If a dynamic buffer is to be used, a flag is checked to see whether the buffer comes from STA or from the requester JCA. A dynamic buffer of the size specified in the RDBMAX field of the RDB is then allocated via the nucleus routine NFGTA.

If RPROOT called for this buffering, RPBUF now sets up the buffered request by first building the buffered request overhead (BRO). The BRO is shown in the section of data structure pictures. It includes a pointer to the requester TSB and JSB, the address of the call block in the requester task, a set of flags, and several fields filled during SVC processing.

After the BRO is completed, RPBUF includes as much of the call block as indicated in the RDBBAS field of the RDB. RPBUF then checks to see if expansions to this basic block are to be included. If so, the next several words of the RDB indicate where to buffer the information (table area or JCA), how much to buffer, and at which offset into the buffered information to place the new information.

If the buffering request is for revision of a partially buffered block, RPBUF copies the BRO and the basic request block from the partially buffered block to the newly acquired block. The old block of memory is released via the nucleus function NFRTA, and expansions are treated like those in buffering for RPROOT.

6.5.3 Dequeuing and Unbuffering Routine (RPDQUE).

When a task is to be scheduled for execution, the scheduler examines the TSB to see if any SVC requests are to be unbuffered to the requester task. If so, RPDQUE is called to remove all queued SVC requests. RPDQUE works with each queued request, returning information from the buffered request block to the requester task. It passes back the return code byte and then uses an RIB to pass back any other information, if the RIB is defined. RPDQUE returns the number of bytes specified in RIBLEN from the offset RIBOFF in the BRB to the offset RIBOFF in the requester call block. Several sets of paired specifications may be present, terminated by pair of zeros. When these pairs are completed, a postprocessor is called, if one is specified in the

RIBPRO field. When unbuffering is complete, RPDQUE releases the buffer via the nucleus routine NFRTA and returns to its caller.

A Wait for Event SVC requires special handling by RPDQUE, which checks the requester task TSB to see which event flags have completed. The flags being tested in the Wait for Event block are then matched against those in the TSB to generate a correct reply in the requester task area.

6.5.4 Other Request Processor Support Routines.

RPMAP2

This routine is part of the DNOS root. It is used by RPROOT to access a processor in the second SVC map file. RPMAP2 adjusts global pointer CURMAP, loads the second map file, and transfers control to the processor routine. If the processor returns to RPROOT, it passes back through RPMAP2, restoring the original map file.

RPPRCK

This routine checks the memory-protection attributes of a portion of memory. It first examines the protection bit in the status word of the task. If protection is enabled, RPPRCK then checks to see if the map register limit indicates write protection. If so, an error is returned. To allow unbuffering of SVC request results, write protection must not be enabled; thus, the error causes the task to terminate.

RPSGCK

The requester call block must be mapped in by a single base and limit register pair to simplify processing. RPSGCK verifies this condition. Given any address and length, RPSGCK uses the relevant map file to ensure that the block addressed is correctly mapped. If not, an error is returned, which may cause the task to terminate.

6.5.5 DNOS SVCs and Processors.

Table 6-2 shows the processors for each of the system-defined SVC opcodes for DNOS. In some cases, a preprocessor is shown, since that module is the one accessed from RPROOT; it may in turn call one of several processors. Some small processors that perform related functions have been collected into a single module; the listing shows both the module name and the processor name for these processors.

Table 6-2 SVC Processors and Modules

NOTATION:	MEANING
A	Alignment on word boundary required
I	May be initiated with SVC 41
(Not Supported)	This SVC code is intentionally omitted.
(pre)	This is a preprocessor
P	Software privileged task required
(P)	Some of the set require software privilege
S	System task required
(S)	Some of the set require a system task
(task)	This processor runs as a task
[nn]	Name of module containing processor

			Processor/Preprocessor
SVC #	Name	Notes	[Module if Different]
00	I/O Operations	A,I,(P)	IOPREP (pre)
01	Wait for I/O	A	RPWT01 [RPWTIO]
02	Time Delay	A	RPTDLY [RPGSVC]
03	Get Date and Time	A	RPGDT [RPGSVC]
04	End of Task		RPENDT [RPPSVC]
05	(Not Supported)		
06	Suspend Task		RPUNCW [RPGSVC]
07	Activate Suspended Task		RPAST [RPGSVC]
08	(Not Supported)		
09	Extend Time Slice		RPETS [RPGSVC]
0A	Convert Binary to Decimal		RPCBDA [RPCONV]
0B	Convert Decimal to Binary		RPCDAB [RPCONV]
0C	Convert Binary to Hexadecimal		RPCBHA [RPCONV]
0D	Convert Hexadecimal to Binary		RPCHAB [RPCONV]
0E	Activate Time-Delayed Task		RPATDL [RPGSVC]
0F	Abort I/O Request by LUNO		IOABRT
10	Get Common Data Address		PMGRCM
11	Change Task Priority		RPCTP [RPGSVC]
12	Get Memory	A	PMGRMM
13	Release Memory	A	PMGRMM
14	Load Overlay	A	PMOVYL (task)
15	(Not Supported)		
16	(Not Supported)		
17	Get Task Bid Parameters	A	RPGTBP [RPGSVC]
18	(Not Supported)		
19	(Not Supported)		
1A	(Not Supported)		

Table 6-2 SVC Processors and Modules (Continued)

SVC #	Name	Notes	[Module if Different]
1B	Return Common Data Address		PMGRCM
1C	Put Data	A	PMGDAT
1D	Get Data	A	PMGDAT
1E	(Not Supported)		
1F	Scheduled Bid Task	A	RPXSBT [RPPSVC]
20	Install Disk Volume	A,P	RPVOL (task)
21	System Log Queue Request	A	LGSVC
22	Disk Management	A,S	DMTASK (task)
23	(Not Supported)		
24	Suspend for Queue Input	S	RPQSUS [RPPSVC]
25	Install Task	A,P	PMPINS (task)
26	Install Procedure/Segment	A,P	PMPINS (task)
27	Install Overlay	A,P	PMPINS (task)
28	Delete Task	A,P	PMPDEL (task)
29	Delete Procedure/Segment	A,P	PMPDEL (task)
2A	Delete Overlay	A,P	PMPDEL (task)
2B	Bid Task	A	RPXTSK [RPPSVC]
2C	Read/Write TSB	A,P	PMRWTB
2D	Read/Write Task	A,P	PMRWTK (task)
2E	Self Identification	A	RPGSID [RPGSVC]
2F	Get End Action Status	A	RPGEAS [RPGSVC]
30	(Not Supported)		
31	Map Program Name to ID	A	PMPMAP (task)
32	(Not Supported)		
33	Kill Task	A	RPKILT [RPPSVC]
34	Unload Disk Volume	A,P	RPVOL (task)
35	Poll Status of Task		RPPTS [RPGSVC]
36	Wait for Multiple I/O		RPWT36 [RPWTIO]
37	Assign Program File Space	A,P	PMPASP (task)
38	Initialize New Disk Volume	A,P	RPINV (task)
39	(Not Supported)		
3A	(Not Supported)		
3B	Set Date and Time	A	RPIDT [RPGSVC]
3C	(Not Supported)		
3D	Semaphore Operations	A,I	PMSEMA
3E	Reset End Action Status		RPREA [RPGSVC]
3F	Retrieve System Data	A	RPRETR
40	Segment Management	A,(S)	SMPREP (pre)
41	Initiate Event	A	RPROOT
42	Wait for Event	A	RPWAIT
43	Name Management	A	NMPREP (pre)
44	Reserved		
45	Get Encrypted Value	A	SECRYP
46	Get Decrypted Value	A	SECRYP
47	Log Accounting Entry	A	PMACCT
48	Job Management	A	JMPREP (pre)

Table 6-2 SVC Processors and Modules (Continued)

SVC #	Name	Notes	[Module if Different]
49	Get Accounting Info from TSB	A	PMACCT
4A	Modify BTA or JCA Size	A,P	PMSBUF (task)
4B	Halt/Resume Task	A,P	PMHALT
4C	Return Code Processor	A	RPRCP (task)
4D	(Not Supported)		
4E	Comm I/O		
4F	Post Event	A	RPPEVT
50	DNOS Performance Functions		
80+	User-defined SVCs		

6.6 USER-WRITTEN SVC PROCESSORS

The standard set of SVCs uses operation codes that range from >0 through >7F. The user may implement SVCs using codes from >80 through >FF. One or more codes may be specified, using any codes within the user-defined range. The user must design the SVC block, build an RDB to describe buffering, build an RIB if information is to be unbuffered, and set up a module of information with the IDT name RPUDAT. During sysgen, the user supplies a file name for the module containing RPUDAT object and ensures that object modules for the SVC processor(s) are in the directory .S$SGU$.USERSVC of the data disk.

6.6.1 User SVC Table.

The user specifies the RDB and RIB information, as well as a set of general information about all SVCs being defined, in a module of tables that contains the following:

* An IDT name of RPUDAT

* DEF statements for RPUMAX and RPUTAB

* REF statements for each SVC processor entry point

* A byte named RPUMAX with a value of the largest user-defined SVC code

* A table named RPUTAB with a two-word entry for each SVC code in the range >80 through RPUMAX

* An RDB for each user-defined SVC code

* An RIB for each user-defined SVC that must return information to the caller

The entries in the table RPUTAB consist of two words each. The first word is the value >E000 and the second word is the address of the RDB for the SVC code being defined. The first entry in the table is for SVC code >80. Each successive entry is for the next sequential SVC code. If a particular code is not defined in the system being generated, the entry in RPUTAB must consist of two words of zero. Figure 6-3 includes the format of RPUTAB when the user is defining several SVCs.

An RDB for a user-defined SVC includes the address of the SVC processor, flags showing how to copy the call block for processing, and the address of an RIB used to return information to the calling task. Table 6-3 shows the format of an RDB.

Table 6-3 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 call block to be copied by the operating system
Byte	Zero
Word	Zero

Figure 6-3 shows several RDB definitions for user-defined SVCs.

An RIB is used by the operating system to return data from the system copy of the call block to the task that issued the SVC. If only the error byte of the call block must be returned, no RIB is needed. If any other information is to be returned, an RIB must be specified in the RPUDAT module. Table 6-4 shows the format of an RIB for a user-defined SVC. The pair of byte fields may be repeated if information is to be returned from several noncontiguous areas in the call block.

Table 6-4 Return Information Block (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

The RPUDAT module must be assembled and its object module pathname must be supplied during sysgen in response to the question about the user SVC table. Figure 6-3 shows a source module for defining two user SVCs, using SVC opcodes >80 and >82. Assume that there is some legitimate reason to omit opcode >81.

```
*------
* 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 6-3 Format of RPUDAT Module

6.6.2 Processors for User-Written SVCs.

The SVC processor must define (DEF) its own entry point. It needs to use SPUSH 1 on entry to save R1 and SPOP 1 to return to the OS. 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:

* R1 - Points to the system copy of the requesting call block

* R4 - Points to the requester TSB

* R5 - Points to the requester saved map file

* R10 - Points to an internal operating system stack

* R13 - The requesting task workspace pointer (WP)

* R14 - The requesting task program counter (PC)

* R15 - The requesting task status register (ST)

The SVC processor must not alter registers 13, 14, and 15. Register 10 should be used only for pushing and popping items on the stack.

Register 1 points to the system copy of the requester's call block. The processor usually gathers all of 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 by using long distance instructions with register 5 as the map file pointer.

The call block as received by the processor has several words of overhead as detailed in the buffered request overhead (BRO) template. This overhead includes the requester's TSB address, JSB address, call block address, and several other pieces of information. Each of these is accessible using negative offsets from the buffered call block pointer in register 1.

When the processor finishes its work, it must return to the operating system by issuing the instruction SPOP 1. The operating system returns information as specified in the RIB for the SVC performed. Control is then passed back to the task that issued the SVC. The DNOS Systems Programmer's Guide includes an example of a user-written SVC processor.

SECTION 7

SEGMENT MANAGEMENT

7.1 OVERVIEW

The segment management subsystem enables tasks to dynamically change the segment set mapped by the task. Segment management also enables a task to guarantee accessibility to a segment until it is no longer needed. Finally, segment management enables a task to write segments to disk if their attributes allow this function.

Segment management also provides the operating system with mechanisms to manipulate data structures even when they are not contained in the same address space. Since DNOS is a job-oriented operating system, system data structures whose scopes are contained within a job are located in separate segments. Thus, the operating system is able to service job-level requests by mapping only the job-level system data structures.

Segment management enables the file management subsystem to manage file buffers. By treating file buffers as segments, file management is able to access any buffer, whether in memory or on disk.

7.2 ARCHITECTURE OF SEGMENT MANAGEMENT

The Segment Manager is implemented as three distinct levels of support. The first level contains routines for mapping the various table areas (JCAs and special table areas), finding Segment Status Blocks (SSB) for specific segments, creating and deleting SSBs and Segment Group Blocks (SGB), and causing segments to be loaded into memory. These routines reside in the system root and are described in the section on nucleus functions.

The second level of segment management consists of SVC processors. These processors reside in the second SVC processor segment of map file 0. This level consists of an SVC preprocessor and several SVC processors. These processors enable user and system tasks to dynamically change the address spaces of their tasks.

The third level of segment management is task-level support that enables the Segment Manager to read a program file directory entry for a segment. This support is needed when a Change Segment SVC is executed on a program file segment whose SSB is not in memory. The program file directory is read to get the segment attributes, length, load address, image record number, and attached procedure IDs (task segment). The task loader contains this support. A special interface is used between the task loader and the segment management SVC processors to perform the segment change after the directory is read.

7.3 SEGMENT MANAGEMENT DATA STRUCTURES

Program files are used to support segment management. A program segment entry is located in the procedure section of the program file, thus limiting the total number of procedures and program segments in a program file to 255. The Install Program Segment SVC builds a segment entry. The format is shown as the program file directory index entry (PFI) in the section on data structure details.

The SGB is the in-memory anchor for a set of segments. The SGB resides in the segment management table area. The FCB points to the SGB for a file. If all segments of a group cannot be contained in the same table area, an overflow SGB is created in a different table area and the SGB points to it.

Each segment group consists of one or more segments. Each segment is described by an SSB, which is allocated in the segment management table area. Special table area SSB′s are in STA. An SSB is created by Segment Manager when a task requests a segment that does not currently exist.

The overhead beet (OVB) is used to contain information about a segment when it is in memory. The OVB is located in the beet (32 bytes) preceding the segment.

The reserved segment table (RST) contains a list of segments reserved by a job. The job information table (JIT) contains a pointer to the RST chain, which resides in the JCA. The RST is built when the first Reserve Segment SVC is done or when the current RST overflows. The RST is deleted when it contains no more segment entries or when the job terminates (after releasing all of the segments). The format of the RST is shown in the section on data structure pictures.

A Set Exclusive Use operation creates an Owned Segment Entry (OSE) which points to the owned segment. The SSB points to an Segment Owner Block (SOB) which points back to the TSB of the task that has exclusive use of it. A Load Segment operation creates a Load Segment Entry (LSE) which points to the segment to be loaded. OSEs and LSEs are chained off the TSB in the JCA. SOBs are allocated in the segment management table area.

The segment management SVC block is shown in the section on data structure pictures as the SMR structure.

7.4 SEGMENT MANAGEMENT ROUTINES

Segment management SVC processing begins in the preprocessor routine SMPREP. Depending on which subopcode is specified, control then is transferred to the appropriate subopcode processor.

7.4.1 SVC Preprocessor (SMPREP).

SMPREP receives control from the SVC decoder, RPROOT, with the pointer to the SVC call block in the task as input. SMPREP verifies that the following conditions are met:

* All of the call block is within the task.

* The subopcode is within range.

* If the operation is a Change or Create Segment, the I/O count and the initiate count for the task are zero, unless the task is software privileged.

NOTE

> The OS does not provide general support for proper completion of I/O when the call block or buffer is mapped out of the task. When DNOS unbuffers the data of an IPC read-type operation, it does not use the task map file, so mapping out IPC read or master read buffers is supported.

* If a LUNO is specified, it is assigned to a file of a valid type and in certain cases, is open.

SMPREP uses a pointer in the Logical Device Table (LDT) for the specified LUNO to determine the File Descriptor Packet (FDP) that contains the File Management Table and File Control Block (FMT,FCB) pair. The FMT,FCB addresses are saved in the segment management SVC block. If the memory-based segment group is specified, an FMT,FCB address of zero is used. The FMT,FCB pair is used to identify the segment group in which the requested segment resides. If the operation is not change or create segment, the SMT,SSB pair for the specified segment is obtained. If it cannot be found, an error is returned. Figure 7-1 shows the overall flow of control to and from SMPREP.

```
                      +-----------+
              XOP     | REQUESTER |
        -----------|   TASK     |
        |             +-----------+
        V
    +---------+
    | RPROOT |
    +---------+
        |
        V
    +---------+
    | SMPREP |        SMLOAD-Load a Segment
    +---------+        SMUNLD-Unload a Segment
        | (BL)         SMEXCU-Set Exclusive Use of a Segment
        V              SMREXC-Reset Exclusive Use of a Segment
   +-----------+       SMCHGS-Change Segment
   |    SVC    |       SMCRES-Create Segment
   |PROCESSORS|        SMRSVE-Reserve Segment
   +-----------+       SMRLSE-Release a Reserved Segment
        |              SMCHKS-Check Segment Status
        |              SMFWRS-Forced Write Segment
        |              SMJRLS-Job Manager Release Segment
        |              SMMDFY-Set/Reset Modified and Releasable
        |              SMBIAS-Bias Segment Address Within Task
        V
  See SVC Processing Description
 for interface to return to user.
```

Figure 7-1 Flow of Control in Segment Manager

7.4.2 Change Segment Processor (SMCHGS).

The Change Segment operation enables a task to change the segment set that comprises its logical address space. The caller specifies either the LUNO for the file in which the segment resides or a flag to signify a memory-based segment. An ID (installed or run-time) uniquely identifies the new segment. The segment to be mapped out of the task is identified by a run-time ID or a map position number.

The Change Segment processor first decides whether the caller is adding, removing, or changing a segment. If the caller is removing a segment, the last segment of the task is unmapped unless it is a task segment. (This removal constitutes an error.) The routine SMRMVE is called to decrement the count of tasks that currently require the segment to be in memory (the task-in-memory count). When this count goes to zero, the segment can be swapped or released from memory; therefore, SMRMVE is responsible for either caching or releasing the segment. (Refer to the program management section for more details.)

Add Segment and Change Segment processing are essentially the same except that during an add there is no old segment to be removed from the task. The routine SMSRCH is called to search for the requested new segment. If it is found, SMSRCH verifies that it may be used by the requesting task. SMSRCH first calls SMFSID to see if the segment is defined. SMFSID uses the FDP and ID to uniquely identify the segment. If the segment is found, the SSB address is returned. SMSRCH then validates the segment attributes for the task. If a non-task segment is share protected, SMCHUC is called to verify that it is used only by the requesting task before SMCHGS is allowed to map it. If a segment is owned but not by this task, mapping is not allowed. If the segment is replicatable and in use, SMSRCH duplicates the SSB. If SMFSID does not find the segment defined in memory, SMSRCH calls SMBLDS to build an SSB. If the segment is a file management buffer or is memory-based, the SSB can be defined completely. However, if the segment is a program file segment, the program file directory on disk must be read to obtain the segment information. Thus, SMBLDS will place program file segments in the initial load state to be processed by the task loader.

Control is received in SMCHGS with the new SSB address. If the new segment is an initial load segment, control is passed immediately to the routine SMEXIT. Otherwise, certain conditions are checked before the segment change is allowed. The task must fit into user memory with the new segment, and the task must not map more than 64K bytes. Also, if any segment other than the last one in the task is being changed, the new segment must be the same size as the old segment. An exception to this rule is made for system tasks, which may change in different-sized segments; however, the segments' logical starting addresses do not change. If these conditions are met, the old segment is removed from the task address space. SMRMVE disposes of it accordingly (not required when adding a segment). SMEXIT is then called to map the new segment.

The routine SMEXIT is responsible for incrementing the use count in the SSB, updating the WCS bit in the status register, building the limit register for the new segment, and updating the protection bits in the limit register. SMEXIT decides whether the new segment is in memory. If not, the calling task is deactivated and suspended while waiting for memory. If the new segment is in memory, its task-in-memory count is incremented, the map base value is calculated, and the calling task is placed into execution with the new segment in its address space.

When an initial load segment is processed, SMEXIT suspends the task on the WOM list. This places the task loader into execution and determines that an initial load segment is being requested (TSBSBN is nonzero). The task loader then tests the SSBs to see if the task is in the initial load state. If so the program file directory entry for the segment is read and the SSB fields are initialized. Now that a segment SSB with the specified ID exists in memory, the task loader calls SMCHGS via an interface routine, PMSMIR. SMCHGS processes the Change Segment as usual except that control is returned to the task loader from SMEXIT (instead of suspending the calling task or placing it into execution). The task loader then loads the task as usual. Figure 7-2 shows the flow of control through the Change Segment processor.

```
        +----------+
        |  SMCHGS  |
        +----------+
            (BL)
             +-----------------------------+
             V                             V
        +----------+                 +----------+
        |  SMSRCH  |                 |  SMRMVE  |
        +----------+                 +----------+
            (BL)                          |
             +--------------+            (B)
             V       |      V             V
        +----------+ |+----------+   +----------+
        |  SMFSID  | ||  SMBLDS  |   |  SMEXIT  |
        +----------+ |+----------+   +----------+
                     V                    |
                 +----------+             +--------------+
                 |  SMCHUC  |            (B)           (B)
                 +----------+        +----------+  +----------+
                                     |  NFTRTN  |  |  NFSRTN  |
                                     +----------+  +----------+
```

Figure 7-2 Flow of Control in Change Segment

Figure 7-3 shows the flow of control if an initial load segment is being accessed.

```
     +----------+
     |  SMCHGS  |
     +----------+
         (BL)
          +-----------------------------+
          |                            (B)
          V                             V
     +----------+                 +----------+
     |  SMSRCH  |                 |  SMEXIT  |
     +----------+                 +----------+
         (BL)                          |
          +--------------+            (B)
          V              V             V
     +----------+  +----------+   +----------+
     |  SMFSID  |  |  SMBLDS  |   |  NFSRTN  |
     +----------+  +----------+   +----------+

     TASK LOADER ACTIVATED WHEN ENTRY PLACED ON
                 ITS QUEUE BY SMEXIT

     +----------+
     |   TASK   |
     |  LOADER  |
     +----------+
        (BLWP)
          +---------------
          V              V
     +----------+   +----------+
     |  NFMAPO  |   |   LOAD   |
     +----------+   |   TASK   |
         (BL)       +----------+
          V
     +----------+
     |  PMSMIR  |
     +----------+
         (BL)
          V
EXECUTE SMCHGS EXCEPT THAT
SMEXIT WILL RETURN TO PMSMIR
```

Figure 7-3 Flow of Control During Initial Load

7.4.3 Create Segment Processor (SMCRES).

The Create Segment operation enables a task to create an empty segment of a certain size with specific attributes. Two types of segments may be created: relative record segments and memory-based segments.

When relative record segments are created, the segment length is the physical record size of the file (obtained from the FCB). The length and attributes for memory-based segments are defined in the call block. Default attributes are readable, nonsystem, disk resident, nonreplicatable, non-WCS, reusable, and noncopyable, though the user may set or reset the execute-protect and share-protect attributes through the call block. The write-protect and updatable attributes are set based on the file protection flags.

The Create Segment processor first decides whether the request is to add an empty segment or to change one. Much of the same validation is required here as in Change Segment to ensure that the new segment can be mapped by the calling task. If the specified conditions are met, SMBLDS is called to build the SSB for the empty segment. An empty segment flag is set in the SSB to inform the task loader that the segment does not reside on disk. If a segment is not being added, SMRMVE is called to dispose of the old segment. SMEXIT is called to finish processing before returning control to the calling task. The task is suspended by SMEXIT since the empty segment is not in memory at this time.

Special processing is required by Create Segment for relative record segments. Before a new SSB is built, a check is made to see if a segment with the same ID already exists in memory. If so, an error is returned. Figure 7-4 shows the flow of control through the Create Segment processor.

```
+----------+
|  SMCRES  |
+----------+
    (BL)
     +-----------+------------+------------+
     |           |            |           (B)
     V           V            V            V
+----------+ +----------+ +----------+ +----------+
|  SMFSID  | |  SMBLDS  | |  SMRMVE  | |  SMEXIT  |
+----------+ +----------+ +----------+ +----------+
   (FILE                                   (B)
   BUFFERS                                  +
    ONLY)                                   |
                                            V
                                       +----------+
                                       |  NFSRTN  |
                                       +----------+
```

Figure 7-4 Flow of Control in Create Segment

7.4.4 Reserve Segment Processor (SMRSVE).

The Reserve Segment operation enables a task to maintain access to a nonupdatable segment when needed, even though the segment is not in any task's address space. Since segments which are not memory-resident may be released from memory when they are no longer in use, this operation is needed to retain access to these segments. The segment is reserved at the job level until a Release Reserved Segment operation is executed or the job terminates. All segments reserved by tasks within a job are contained in the RST to which the JIT points. Segments are removed from the RST whenever a Release Reserved Segment operation is done. When the job terminates and the RST is not empty, the job management subsystem is responsible for releasing the remaining segments. Reserved segments are swapped if their memory is needed. The SSB for the reserved segment remains in memory as long as the segment is reserved.

SMRSVE searches the RST chain for a free entry to contain this segment's run-time ID. If no free entries exist, a new RST is built. The reserve count in the SSB is incremented. Control is then returned to the calling task via the Request Processing subsystem.

7.4.5 Release Reserved Segment Processor (SMRLSE).

The Release Reserved Segment operation is used to release a segment that has previously been reserved within the job. The RST includes an entry for the segment if it was reserved in the job. The processor returns an error if an entry is not found; in effect, a job cannot release segments it has not reserved.

SMRLSE first decides if the requested segment was reserved by the job. If so, the entry is deleted from the RST. If the RST is empty, it is delinked from the RST chain and deleted. The SMT, SSB pair is used to find the segment that is being released. The reserve count is decremented. If the segment is no longer in use or reserved, the segment is left cached or is deleted from memory. SMDSSB does the following processing. If the segment is in memory and is reusable, the segment remains cached in memory (unless the releasable flag is set in the SSB, in which case the segment is deleted). If the segment is in memory but is not reusable, the segment is queued for deleting by the task loader, and the SSB is deleted. If the segment is not in memory, the swap table entry for the segment is deleted along with the SSB. Control then returns to the calling task via the request processing subsystem.

7.4.6 Check Segment Status Processor (SMCHKS).

The Check Segment Status processor returns information about a certain segment. Such information includes the segment run-time and installed IDs, length, attributes, whether the segment is a task and whether the segment is memory-based. If the segment is mapped by the task, the logical address of the segment is returned. The segment need not be mapped or reserved by the task requesting the status.

If the segment is mapped by the task, the status information is returned along with the logical address. If the segment is not found in the task, the segment group is searched. If the specified ID is found, the status information for the segment is returned.

7.4.7 Forced Write Segment Processor (SMFWRS).

The Forced Write Segment processor writes a segment to its home file position (if updatable and modified). The segment is represented by an SMT and SSB. The task requesting the write is suspended until completion of the write. The disk I/O is accomplished by a dedicated queue server of the write queue, PMWRIT.

If the segment does not exist, an error is returned. If the segment exists, a check is made to determine if it is updateable. If not, an error is returned. If it is updateable, is in memory, and is modified, the write will occur. The OVB for the segment is queued to the write queue. The calling task is then suspended until completion of the write.

PMWRIT is activated whenever an entry is placed on the write queue. PMWRIT calls the file management routine FMIO to write the segment to its home file. It determines whether the segment is a program file or data file segment. If it is a program file segment, the home file record number is contained in the SSB word SSBREC. For a data file segment, this record number is contained in the installed ID field of the SSB. After FMIO is called to perform the disk I/O, SMDSSB is called. Finally, if a task is suspended for the write (that is, the OVB points to a forced write call block through the OVBBRB field), the task is placed back into execution via NFEOBR. Figure 7-5 is a diagram of the flow of control through the Forced Write processor and task.

```
+----------+
|  SMFWRS  |
+----------+
    (BL)
     +-----------------------------+
     |                            (B)
     V                             V
+----------+                  +----------+
|  NFQOVB  |                  |  NFSRTN  |
+----------+                  +----------+

FORCED WRITE TASK(PMWRIT) ACTIVATED WHEN ENTRY PLACED ON
         ON ITS QUEUE BY SMFWRS

+----------+
|  PMWRIT  |
+----------+
    (BL)
     +---------------
     V       |      V (THROUGH NFMAPO INTERFACE)
+----------+ | +----------+
|   FMIO   | | |  NFEOBR  |
+----------+ | +----------+
             |      |
         +--------+ - > PLACES CALLING TASK BACK
         | SMDSSB |     INTO EXECUTION
         +--------+
```

Figure 7-5 Flow of Control in Forced Write

7.4.8 Release Job Segments Processor (SMJRLS).

This operation is used by job management to release reserved segments in a specified job when the Job Manager is terminating that job. This operation may be executed only by a system task (specifically Job Manager).

SMJRLS is called with the SMT,SSB address and the JSB of the terminating job. The JCA of the terminating job is mapped in for the segment to be released. SMRLSE is called to process the Release Segment operation as usual. SMRLSE then returns control to SMJRLS, which returns control to the caller via the request processing subsystem. Figure 7-6 shows the flow of control through the Release Job Segments processor.

```
                 +----------+
                 |  SMJRLS  |
                 +----------+
                    (BL)
         +-------------+-----------------------------+
         |             |                             V
         V             V                    RETURN TO TASK VIA
    +----------+ +----------+               REQUEST PROCESSING
    |  SMMJCA  | |  SMRLSE  |                    SUBSYSTEM
    +----------+ +----------+
```

Figure 7-6 Flow of Control in Release Job Segments

7.4.9 Set/Reset Modified and Releasable (SMMDFY).

The Set/Reset Modified and Releasable operation is used to mark a segment of a task as releasable or nonreleasable and to mark an updatable segment as modified or not modified. The default conditions for segments are nonreleasable and not modified.

The SSB of the segment is located, and the flags for the releasable and modified states are set according to the SVC request.

7.4.10 Bias Segment Address Within Task (SMBIAS).

The Bias Segment Address Within Task operation is used to position segment two or three of a task at a new logical address. This is used primarily by the System Configuration Utility. This subopcode (>08) is not available to users.

7.4.11 Set Exclusive Use of a Segment (SMEXCU).

The Set Exclusive Use of a Segment operation is used to extend the share-protection attribute to segments not currently mapped in by a task. A segment which has had exclusive use set is said to be an owned segment. Other users who try to map in the owned segment will get a shared segment violation error (unless it is replicatable, in which case a replicated copy will be mapped). The set operation also has the functionality of a reserve segment operation. That is, even if an owned segment has use and reserve counts of zero, the segment will not be deallocated.

If the operation is to succeed, the following conditions must be met:

* The segment must not currently be owned by either the task issuing the SVC or another task.

* The segment must not be in use by any task but the issuing task.

SMEXCU calls SMCHUC (Segment Management Check Use Count) to perform this function. Exclusive use of special table areas (SMTs, FMTs, PBMs) is not allowed. Once it is determined that the preceding conditions are met, a segment owner block (SOB) is linked to the SSB, indicating which task owns this segment. An owned segment entry (OSE) is linked to the issuing task's TSB, indicating which segments the task owns.

7.4.12 Reset Exclusive Use of a Segment (SMREXC).

The Reset Exclusive Use of a Segment operation relinquishes a task's ownership of a segment. The operation will succeed only if the segment is currently owned by the task issuing the SVC. The SOB is delinked from the SSB and its memory released. The OSE is removed from the list of owned segments linked to the TSB and its memory released. If the segment is not in use or reserved, it is deleted.

7.4.13 Load a Segment (SMLOAD).

The Load Segment operation assures the user that the specified segment will be in memory while the task that issued the SVC is executing. The segment will not be mapped into the task address space. A segment may be loaded by more than one task regardless of its attributes. When loading a segment, a load segment entry (LSE) is built and attached to the loading task's TSB.

SMLOAD is not only an SVC processor but is accessed with a BL interface by nucleus routines. It executes in Map 0.

7.4.14 Unload a Segment (SMUNLD).

The Unload Segment operation detaches the segment from the task so the segment does not need to be in memory when the task is in memory. An error is returned if the segment was not loaded by the task. The LSE is delinked from the TSB. If the reserve, use and exclusive use counts are zero, the segment may be cached or deleted.

SMUNLD is not only an SVC processor but is accessed with a BL interface by nucleus routines. It executes in Map 0.

7.5 SEGMENT MANAGEMENT TABLE AREA

Segment Manager maintains its internal data structures in special table areas that are separate from the STA. These blocks contain SSBs and SGBs. During sysgen, a variable number (one or more) of these areas are defined to fit into the second segment of the system mapping scheme (replaces JCA segment).

Sysgen creates an SSB in the STA for each segment management table area. These SSBs are used by the Segment Manager to access each table area. The tables reside in the memory-based segment group; thus, a memory-based SGB resides in the STA. Each table area has the standard memory management overhead along with a pointer to the first SGB in the table and information required to generate run-time IDs for SSBs. The Get and Release table area routines (NFGTA and NFRTA, respectively) are used to allocate memory in the special table areas.

Whenever a new segment group is being created, Segment Manager decides which table area has the most unused memory and allocates the segment group into this area. Segment Manager will attempt to allocate all segments of a group within the same table area. If this is not possible, an overflow SGB is created in a different table area. The overflow SGB contains the same information as the SGB (which points to the overflow SGB). Thus, Segment Manager can search all segments of a group by searching the segments that reside in the table, then search the segments that reside in a table to which the overflow SGB points. Figure 7-7 is a general diagram of the Segment Manager table scheme (given two table areas).

```
+---------------------------------------------+
|       Contains Memory-based SGB and SSBs    |
|       for special table areas, ROOT and     |
|  +-----+          COMMON.                   | STA
|  | SGB |--+                                 |
|  +-----+  |   +-----+   +-----+             |
|     |      +-->| SSB |-->| SSB |-...-->0     |
|     |          +-----+   +-----+             |
+-----|---------------------------------------+
      |
      | Overflow SGB pointer
+-----|---------------------------------------+
|     V                                       |
|  +-----+                                    |
|  | SGB |--+  +-----+   +-----+              |
|  +-----+  +->| SSB |-->| SSB |-...-->0      |Special table #1
|              +-----+   +-----+              |
|  +-----+                                    |
|  | SGB |--+  +-----+   +-----+              |
|  +-----+  +->| SSB |-->| SSB |-...-->0      |
|     |        +-----+   +-----+              |
+-----|---------------------------------------+
      |
      | overflow SGB pointer
+-----|---------------------------------------+
|     V                                       |
|  +-----+                                    |
|  | SGB |--+  +-----+   +-----+              |
|  +-----+  +->| SSB |-->| SSB |-...-->0      |Special table #2
|              +-----+   +-----+              |
.                                             .
                  ETC.
.                                             .
+---------------------------------------------+
```

Figure 7-7 Segment Manager Table Organization

SECTION 8

JOB MANAGEMENT

8.1 JOB CONSTRUCT

A job is the fundamental work unit to which DNOS logical resources are allocated. These resources include files, devices, IPC channels, and environments of names.

The goals of the job construct in DNOS are the following:

* To provide a structure for the information about a group of related tasks (for example, resources allocated, security level, and accounting information)

* To provide the capability of divorcing tasks from an active physical terminal

* To provide a vehicle for easy migration of applications between DNOS configurations by isolating a set of tasks from all others in a system

A job consists of one or more tasks, a set of job-local variables, a set of resources, a set of job-local LUNOs, and a job ID. The operating system constitutes a job in that it owns files, devices, and channels and consists of a group of coöperating tasks.

A job has an associated priority. This priority is used for scheduling various system services. Such as disk events and positioning requests into the spooler queue.

Management of resource allocation by jobs in DNOS provides a level of isolation between different jobs. Once resources have been allocated to a job, the execution of the job can be independent of the existence of other jobs. Hence, jobs also provide a migration vehicle from a single- to a multiple-application environment.

8.2 OVERVIEW OF JOB MANAGEMENT

The Job Manager assigns and manages job identifiers, limits the number of jobs in the system, and provides system access security. To support these functions, the Job Manager processes the following SVCs:

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

A task requests creation of a job via the Create Job operation of the Job Management SVC (>48). The Job Manager performs security checks to validate the integrity of the request and generates a unique job ID. The job is created and is set into execution if it will not exceed the system job limit. If this is a batch job, it must not exceed the background job limit also. If either limit will be exceeded, the job is placed on a queue, waiting for some other job to terminate. Security on the Halt, Resume, Kill, Get Job Information, and Modify Job Priority operations is provided so that only a user with the same user ID or a part of the system job may perform these operations. A job determines its own job ID through the use of the Self Identification SVC (>2E). Status information on jobs is obtained via system utilities.

8.3 ARCHITECTURE OF JOB MANAGEMENT

Job Manager is a system task that executes in the system job. It is coded in Pascal with minimal run-time support and is a disk-resident queue server. It has an assembly language initialization routine, which contains the stack space for the Pascal routines. Like other system tasks written in Pascal, Job Manager uses routines in DSC.PASASM to call nucleus functions.

Job Manager serves a singly linked list of entries. The Job Manager logical address space consists of the system root, a JCA segment, and task code. Any task requesting a Job Management SVC is suspended until the request has completed.

8.4 JOB MANAGEMENT DATA STRUCTURES

Among the data structures used by Job Manager are several structures particular to segment management and nucleus functions. These include SSBs, BRBs, and TSBs. The job-related structures primarily used by Job Manager include the JCA, JSB, and JIT.

The JCA contains all data structures local to a given job. The JCA is allocated from free memory and can be swapped when all the tasks in a job are swapped out of memory. The JCA may be expanded, as necessary, up to the maximum size specified during sysgen.

The JSB carries all global data about the job, including the address of the JCA, job ID, job name, and priority. It is allocated from the STA.

The job management SVC request block is the JMR.

The JIT contains the list headers for structures in the JCA and is allocated as the first portion of the JCA.

Details of each of these structures are shown in the section on data structure pictures.

8.5 JOB STATES

The state of a job is maintained in its JSB and changes only in response to SVCs initiated by the user. The following states are possible:

* Creating - A job is in this state only during the execution of the SVC that creates that job, or while waiting for the active or background job count to drop below this limit.

* Halted - A job is in this state when halted by a Halt Job SVC. Only queue server tasks in this job can continue to execute. Any other tasks that attempt to become active while the job is in this state are suspended.

* Executable - A job in the executable state can have its tasks scheduled for memory and CPU.

* Terminating - A job is placed in this state when it is killed or after its last task has terminated. At this point, no more tasks may be bid in the job, and Job Manager begins releasing all resources and data structures within the job.

- * JCA being expanded - A job is placed in this state while its job communication area is being expanded due to job requirements for more data structures than the current JCA can accommodate.

In addition to these job states, Job Manager can also cause a task to enter the job suspended state. This state is used when halting a job.

8.6 DETAILS OF JOB MANAGER ROUTINES

Job management SVC processing begins in the routine JMPREP. Depending on which operation is requested, control then is transferred to the appropriate operation processor.

8.6.1 Job Manager Preprocessor (JMPREP).

JMPREP is a small assembly language routine that resides in the scheduler segment of map file 0. It causes the BRBs for certain sub-opcodes to be rebuffered to include more information than originally buffered by the SVC decoder. To rebuffer, JMPREP builds an RDB showing what to rebuffer and calls the request processor buffering routine, RPBUF, to perform the data movement. JMPREP then queues the BRB to the Job Manager queue for processing and returns to the scheduler to suspend the requester task.

8.6.2 Job Manager Request Processing Task (JMMAIN).

JMMAIN is the main module for Job Manager. It acquires and releases segments as necessary and initializes local variables. It also provides all functions common to the SVC processors, such as retrieving the BRB and getting the proper JCA. At the end of SVC processing, it will start any jobs on the waiting queue, provided that the job limit and batch job limit are not exceeded.

8.6.3 Create Job Processor (JMC$).

JMC$ must map in the caller's JCA area and retrieve some of the values stored in the JIT. When the new user ID flag is not set in the flag word of the BRB, then the user ID, passcode, account number, and privilege level are copied out of the caller's JCA into the BRB for use at a later time.

Once the call block is buffered, a JSB is built in the STA. A unique job ID is generated and is used to identify the job while it remains in the operating system. This ID is placed in the JSB and is returned to the caller in the BRB. The job priority is checked to see if it is in the range of 0 through 31. If not, the request is returned with an error status. Otherwise, the JSB state is set to indicate that it is being created.

Next, Job Manager obtains memory for the JCA area by executing a Create Segment SVC. This segment is placed in the second map segment of Job Manager, replacing the system JCA segment. The size of this JCA area is specified in the call block as 1, 2, or 3. The code 1 is for the smallest JCA size; the code 3 is the maximum JCA size. The logon default is the medium JCA size. All of the memory management overhead is initialized in the JCA, and segment manager SSB addresses for the JCA are stored in the JSB. The queue headers for the job level queue servers are initialized, and the JCA segment is reserved.

The station ID of the job being bid is next verified to be sure that the station specified exists and is available for use. If an illegal station is specified, the job creation request is denied and an error is returned. If the station is legal, creation continues, with Job Manager assigning a job-local LUNO to DUMY.

The requester may specify a logical name and synonym segment in the SVC block. (The function of this segment is detailed in the section on I/O.) When this field is zero, no action is taken. Otherwise, the segment is checked to see if it is a memory-based segment. When it is located, the segment manager SSB addresses are stored in the JIT, and the segment is included in the job reserved segment list. If the segment is not found or is not memory based, the request to bid the job is aborted and an error is returned to the user.

After the synonym segment is processed, the user ID and passcode are verified. When the new user ID flag is set, all information must be verified before it can be used. The user ID and passcode are kept on disk in a predefined system file. A search is made for this user ID in this file. Once the ID is found, the passcode specified is encrypted and compared against the encrypted passcode on disk. Any error in this process aborts job creation. If the file .S$ACCVAL exists, the account ID is verified against the file entries. If a match is not found, the job is aborted. If the file .S$ACCVAL does not exist, no checking is done.

The final step in creating a job is to bid the initial task. First, the JSB is linked into the system JSB list. Then the parameters for the task are built into a Bid Task SVC, which is issued from Job Manager. (Note that because Job Manager is issuing the SVC, the specified program file LUNO must be global so that it appears in the new job's LUNO hierarchy.) Any error returned from the Bid Task SVC is placed in the BRB and aborts the Create Job SVC.

The new task is initially bid in a halted state. After the bid has been completed, a job initialization entry is placed on the accounting queue and the job is put on a wait queue.

Whenever the Create Job SVC has been aborted, the JCA and JSB must be returned to free memory. The BRB for the call is sent back to the caller, along with the error. A temporary BRB is created to indicate job termination and is placed on the Job Manager queue. This entry is processed by JMD$, releasing all the resources the job had and returning them to the system.

8.6.4 Halt Job Processor (JMHALT).

JMHALT calls the verify routine, JMVRFY, to verify that the specified job ID exists and that the requesting job has the authority to perform the halt. The job state is then checked to ensure that the job is active. If it is not active, an error is returned. Otherwise, the job state is changed to halted. The TSB list for the job is then searched for all active tasks. During this search, the scheduler must be inhibited to prevent any change in the list. When active tasks are found, if they are not queue server tasks they are delinked from the job active list and the task state is changed to indicate that the job is suspended. After all of the active tasks are found, the scheduler is enabled and the BRB is returned to the caller with a successful completion code.

While the job is in a halted state, only queue server tasks in the job can be made active. If any other task tries to become active, the nucleus function NFPACT places the task in the job suspended state.

8.6.5 Resume Job Processor (JMRESU).

After JMRESU calls JMVRFY, if no error was found, it checks the job state to see if it is halted. If it is not halted, an error is returned to the caller. If the job is halted, the job state is changed to executable, and the TSB list is searched for tasks in the job suspended state. When such a task is found, Job Manager calls NFPACT to place the task back on the active list. The scheduler must be inhibited during the TSB search so that the TSB list is not altered. After the search, a successful completion code is placed in the BRB, and the BRB is returned to the requester.

8.6.6 Modify Job Priority Processor (JMPRIO).

JMPRIO calls JMVRFY and, if no error was found, it checks that the caller is the system operator. If not, an error is returned. The new job priority is checked to see if it is within the valid priority range. If not, the request is aborted. Otherwise, the tasks within the job are updated to reflect the new priority. Job Manager calls a nucleus routine to obtain the new task priorities. During this time, the scheduler must be inhibited. After all of the priorities are modified, Job Manager delinks the first active task in the job and then relinks it. This inserts the JSB in the right position for the scheduling queue. The new priorities take effect when Job Manager completes its SVC requests. Job Manager then enables the scheduler and returns the BRB with a successful completion code.

8.6.7 Map Job Name Processor (JMMAP).

The Map Job Name processor searches the JSB list for the job name specified in the BRB block. It returns the job ID of the first job that it finds with the same user ID as the calling task. Jobs in terminating state are not considered. An error is returned if no matching job name is found under that user ID, or if the job name is duplicated. In the latter case, the job ID of the first matching job is returned. The user ID and job names of each job are kept in the JSB (which is memory resident) to avoid excessive swapping during this operation.

8.6.8 Get Job Information Processor (JMINFO).

The Get Job Information processor returns information about the job identified by the job ID in the requestor call block. If an ID of zero is specified, information about the caller's job is returned. JMVRFY is called, and if no error is returned, this processor returns the job name, priority, user ID, account ID, privilege level, and the run ID of the calling task.

8.6.9 Kill Job Processor (JMKILL).

The Kill Job processor terminates jobs within the system. JMKILL verifies that the user has access to the specified job and that the job exists. The job state is then checked to see if the job is already in a terminating state. If it is, an error is returned in the BRB to indicate this condition. When the job is in the create state, it must be deleted from the waiting-to-execute job queue. At this point, the job state is changed to terminating so that no new tasks are bid in this job. The scheduler is then inhibited during the kill process for each task.

Job Manager kills each task by calling the nucleus function NFTERM. The TSB contains a flag to indicate whether end action is allowed on a kill request. If it is not set, the task may take end action but will not be able to reset its own end action bit. A time-out value for end action prevents the task from executing indefinitely. Job Manager returns a successful completion code in the BRB when all of the tasks have been processed through NFTERM.

Job Manager suspends (awaiting queue input) at this point to allow all of the tasks to terminate. The termination processor, PMTERM, places an entry on the Job Manager queue to notify Job Manager that the last task has terminated. When the entry arrives, control is given to JMD$ to complete the job termination process.

8.6.10 Job Clean-Up Routine (JMD$).

JMD$ is a support routine that can be activated only by an aborted Create Job SVC or by PMTERM. The call is made when the last task of a job has terminated. JMD$ is responsible for releasing any attached resources and all memory blocks associated with the job.

JMD$ may be called during the create process to clean up an aborted Create Job SVC. JMD$ first determines if the JCA area for the job exists. Job manager releases the JSB if the JCA was not created.

The first set of operations required for the JCA is to release any job-local LUNOs still assigned. The I/O sub-opcode to Release LUNO in Another Job is used for this purpose. The entire list of LUNOs is searched, and a call block for each of these LUNOs is created and passed on to the I/O Utility (IOU).

After all of the LUNOs have been released, all resources that have been attached by the job are released via calls to IOU for each resource found. The resource list is searched, and for each entry found a Detach by Number SVC sub-opcode is issued.

The next structure to be deleted is the reserved segment list. For each reserved segment, a call is made to the Segment Manager to cancel the reserve. This includes the reserve on the JCA segment. The JCA segment is mapped into Job Manager during this release process and is not released from memory until Job Manager changes its second map segment. All other segments, such as logical name and synonym segments, are released and become eligible for deletion if no other job has reserved them. All clean-up of memory-based segments is accomplished by Segment Manager when Job Manager releases the reserved segments. The table memory for the reserved blocks is also released from the JCA by Segment Manager during this process.

At this point in the clean-up process, the JCA should not have any dynamic memory allocations in it. If no other job has reserved this JCA segment before the job was put in the terminated state, this JCA is deleted as soon as Job Manager releases it from its address space.

The last step in JMD$ is to release the JSB from the STA. The JSB is deleted from the JSB list, and the memory is returned. A job termination entry is placed on the accounting queue and the active job count is decremented.

8.6.11 Verify Job ID Routine (JMVRFY).

The verify routine is used by JMKILL, JMINFO, JMRESU, JMHALT, and JMPRIO to determine if the caller is allowed to execute the SVC in question. This routine searches the JSB chain to find the appropriate job ID. Jobs in terminating state are not considered. If the job ID is not found, an error is placed in the BRB and the request is aborted. Otherwise, JMVRFY checks to see if the operation is being executed on the system job. Any attempt to perform one of these SVCs on the system job receives an error. The last check made is to verify that the requesting task has the privilege to perform the SVC. Valid requesters are the system operator, tasks whose jobs have a flag set to bypass ownership tests, and tasks whose jobs have the same user ID. All other requesters receive errors. On successful completion, JMVRFY returns to the calling routine with the JCA of the requested job mapped into the second map file segment of Job Manager and with the JSB address of the job requested.

8.7 IMPLICATIONS OF JOB BOUNDARIES

In theory, a task running in a job is not aware of any other job. It may interact with the system job by issuing an SVC or it may interact with another job by using a global IPC channel, but it theoretically is independent of and unaware of any other job. In reality, it may be necessary for two tasks in different jobs to communicate with each other.

There are two major mechanisms supported by DNOS for cross-job communication: global IPC channels, and event SVCs. The major difficulty with global channels is that the owner task must be the first task to assign a LUNO to the channel, but, aside from that, they have all the power that the various channel configurations provide. Events provide a more limited capability, being mainly a synchronization feature. Any task which knows another task's job ID and task run-time ID can issue a Post-Event SVC which causes a specified event to complete for the task specified by the job ID and run-time ID. The specified task must issue a Wait for Event SVC for the event number of the expected event. It will then be activated when the Post Event SVC is issued.

SECTION 9

PROGRAM MANAGEMENT

9.1 OVERVIEW

Program management supports task-level requests to execute tasks, load overlays, and terminate tasks. Support is also available to perform sychronization operations, to read and write task memory or TSBs, to get and release user memory, and to get and release system common. Program management includes processors for the full set of SVCs to support program files.

Program management is implemented in three distinct levels. The first level includes support routines that reside in the root of the operating system. These routines are callable by various program management routines and perform specific functions. The second level is the set of processors for the program management SVCs which execute at XOP level. The third level is the processors for program management SVCs that execute as queue serving tasks. Many program management SVCs are implemented in the second level only, but some require a third level (for example, disk I/O may be performed only at the third level).

9.2 DATA STRUCTURES USED BY PROGRAM MANAGEMENT

In addition to JSBs, TSBs, various segment management structures, and the JCA, program management uses a number of lists and queues to coordinate the efforts of its components. These structures include the following:

Waiting-on-Memory (WOM) list
A list of JSBs anchored by the NFPTR field WOMJSB, with each JSB having one or more TSBs for tasks that must be loaded. The TSBs of each job are linked from the JITWOM anchor in the JIT of the JCA. The JSBs are ordered by JSB waiting priority as carried in the JSBWPR field.

Loader Queue
A linked list of OVBs, each representing a block of user memory to be released. The queue is anchored at LDRQUE in NFQHDR.

Cache List
A linked list of OVBs for segments currently in memory but not currently used by any active task. (These are the most recently used segments, and they may be used later or deallocated.) This list is anchored at CHELST in NFDATA.

Active List
A list of JSBs for jobs with tasks ready to execute. This list is anchored in the NFPTR field ACTJSB, organized by priority JSBAPR. The TSBs for the active tasks are linked from the JITACT anchor in the JCA.

Time-ordered List (TOL)
A list of task segments (OVBs) representing all tasks in the system eligible for the active list, ordered by most recently loaded.

Time Delay List
A linked list of time delay entries, anchored by the NFPTR list header TDLHDR. The entries reside in the STA and consist of task identification information and time delay values. Wait for Event SVC's are also linked on this list.

Swap Table
A linked list of swap table entries used to keep track of allocated space and segments on the swap file. Whenever a segment is swapped to the swap file, a swap table entry is built for that segment in the system JCA, and a pointer to it is placed in the SSB. This table is used by the segment swapper task to keep track of which records are allocated in the swap file. The anchor for this list is ROLDIR in PMDATA.

9.3 DETAILS OF PROGRAM MANAGEMENT ROUTINES

Program management routines perform the functions of bidding a task, loading a task for execution, and deallocating resources when a task terminates.

9.3.1 Task Bid Processor (PMTBID).

When a Bid Task SVC (>2B) is executed, the nucleus function NFTBID attempts to add the task to the active list. If NFTBID is unable to bid the task, the SVC block is placed on the task bid queue that places the queue server PMTBID into execution.

PMTBID performs the initial setup for a task. It ensures that the task exists by locating the task segment in memory or by reading the program file directory. Then, the procedure segments (if any) are located in memory, or they are built from the program file directory. The map file limit registers are built for the task and the TSB for the task is placed on the WOM list so that its segments will be loaded by the task loader.

Execution of PMTBID proceeds as follows. The routine PMGSSB is called to find or build an SSB for the task segment. The routine SMSRCH is called to search the program file segment group for the requested segment. If found, it is used by the task being bid (assuming the segment attributes allow this). If the segment is not found, SMBLDS is called to build an initial load SSB for the segment. (For details see the description of segment management.) When control is returned to PMGSSB and if the SSB returned is in the initial load state (that is, the program file directory has not been read for the segment), PMRDIR is called to read the program file directory entry for the segment. PMMPRI is then called to calculate the initial runtime priority. Next, PMTBID calls PMITSB which calls PMGSSB for the attached procedures, sets up the map file limit registers (including protection), determines the total mapped length and memory used, and initializes the task status. A loaded segment entry (LSE) is built for the JCA of the task, so that the JCA will be loaded into memory when the task is loaded. PMTBID calls PMNMGR to inform the name manager that a new task exists in the job, then links the TSB into the TSB tree and activates the task. The task is now ready to be loaded by the task loader. Finally, the calling task, if any, is killed or suspended if such action is requested in the SVC block.

9.3.2 Task Loader (PMTLDR).

PMTLDR loads tasks into memory when they are initially bid, when they have been swapped out of memory and are rescheduled to run, and when a Change Segment or Create Segment operation is done for a segment that is not in memory. The task loader serves the loader queue and the WOM list. Included in the task loader task are the get and release user memory routines and the task swapping routine.

PMTLDR begins to execute whenever an entry is placed on the loader queue or WOM list. The loader queue is processed first. It contains a list of segments whose memory must be deallocated. The return user memory (PMRUM) routine is called for each segment.

After processing the loader queue, PMTLDR checks the WOM list for a task to be loaded. If one is found, PMTLDR attempts to load the task into memory. PMTLDR first checks to see if the task is doing a Change Segment operation which requires initialization of an SSB. If so, PMRDIR and PMSMIR is called to allow the segment manager to complete its processing. PMTLDR calls the routine PMALSG to allocate memory for the task's JCA and later for each segment of the task. The JCA must be in memory before the task can be loaded, since the JCA contains the TSB. PMALSG first checks to see if the segment is already in memory. If so, the segment is removed from the cache list if the task-in-memory count is zero. If the segment is not in memory, the get user memory (PMGUM) routine is called to allocate memory. If memory is available, PMALSG increments the task-in-memory count and returns the segment to PMTLDR. If memory is not available, PMROLL is called to attempt to obtain memory for the segment.

After memory is allocated for all of the task segments, PMLDSG is called to load each segment into memory. If the segment is already in memory or is empty, the load is skipped. Otherwise, the segment is loaded from its home file or roll file. The record number for the segment is computed, and the file management routine FMIO is called to load the segment from disk into memory. After the load is completed, the OVB is initialized and control is returned to PMTLDR.

After all of the task segments are loaded, the map file bias registers are calculated and the task segment is placed on the TOL by calling NFATOL. The task is put on the active list, and PMTLDR continues processing the loader queue and the WOM list.

If insufficient memory is available to load a task, the task loader calls a swapping routine (PMROLL) to attempt to free memory by temporarily writing segments to the swap file. The swapping routine includes two phases. The first phase processes the cache list (segments in memory that are reusable but not currently in use), freeing any available memory. If the first phase does not yield enough memory, the second phase begins processing the TOL to find a task that can be deactivated and swapped.

Processing the cache list involves searching the list, last to first, for available segments. A segment on the cache list can be swapped if its TILINE I/O count is zero. A maximum count for buffer segments and program file segments is maintained to ensure that memory does not become too fragmented by cached segments. (See the roll parameters shown in the NFDATA template in the section on data structure pictures.) PMROLL tries to prevent the rolling of JCAs and, in an attempt to improve performance, tasks doing file I/O and file buffer segments.

When a swap candidate is found, it is queued to the write queue to be written to its home file, written to the swap file, or released, based on its attributes, use count, and whether it was modified. PMROLL returns a code of zero if it was able to free enough memory before returning. However, PMROLL might have queued an entry on the write queue, in which case PMROLL returns a code of 2 to the task loader, indicating that memory will not be available until I/O completes. If PMROLL finds an eligible segment doing file I/O, it returns a code of 4, indicating that task loader should serve the file I/O request first.

If none of these conditions arises after processing the cache list, PMROLL processes the TOL. The TOL is searched, last to first, to find the most eligible task for deactivation and swapping. The following are three categories of tasks on the TOL, listed in the order of most eligible to least eligible for swapping:

1. Tasks suspended for more than a minimum amount of time or suspended while waiting for coroutine activation. The longer the suspension, the more eligible for swapping.

2. Tasks of lower priority than the task being loaded by the task loader. The lower the priority, the more eligible the tasks are for swapping.

3. Tasks that have the same priority as the task being loaded by the task loader and that have executed for a minimum amount of time since they were last loaded. The longer the execution time, the more eligible the tasks are for swapping.

After an eligible task entry is found on the TOL, the task is deactivated and its segments are placed on the cache list if they are not being used by other active tasks. The cache list is then processed again before returning to the task loader with the return code from the cache list processor. If no eligible entries are found on the TOL, a return code of 3 is given to the task loader, indicating that no memory was found to release.

Figure 9-1 shows the flow of control through the task loader.

```
            ENTRY PLACED ON LOADER QUEUE OR WOM LIST
                                |
                                V
                           +----------+
                           |  PMTLDR  |
                           +----------+
                               (BL)
                                |
                                V
           +--------------------+
           |                    |
           V                    |
      +----------+              |
      |  PMRUM   |              |
      +----------+              |
        Loader                  |
         Queue                  |
                                V
                 +-----------------+-----------------------+
                 |   for JCA and   |                       |
                 V   each segment  V                       V
            +----------+      +----------+          +----------+
            |  PMALSG  |      |  PMLDSG  |          |  NFPACT  |
            +----------+      +----------+          +----------+
               (BL)               (BL)                     |
                 |                  |                      V
                 V                  |                 ACTIVATES
          +------+-----+            |                    TASK
        (BL)         (BL)           |
          |            |            |
          V            V            V
     +----------+ +---------+ +----------+
     |  PMGUM   | |  PMROLL | |   FMIO   |
     +----------+ +---------+ +----------+
```

Figure 9-1 Flow of Control in Task Loader

9.3.3 Task Termination Processor (PMTERM).

When a task terminates by issuing an End Task SVC or is killed by another task or by the operating system, the initial processing is done by NFTERM (see the section on nucleus functions). That processing includes placing an entry on the task termination queue, which is served by PMTERM. PMTERM cleans up the memory structures associated with the task (TSB).

If there is no end-action and the task was a task- or job-local channel owner, PMTERM notifies IPC. For each LUNO on the TSBLDT chain PMTERM issues an abort request. If no end-action is specified, PMTERM closes the LUNO and waits for I/O to complete.

Then PMTERM either waits for or causes all other outstanding requests to complete and if end-action is specified, the task is activated. Otherwise, PMTERM releases all LUNOs, logs a task error (if necessary), informs the name manager that a task is terminating, and delinks the TSB from the TSB tree. This may involve killing descendant tasks or activating a parent task. If there is only one task left in the job (file manager), it is killed. If there are no more tasks left, job manager is activated to terminate the job.

9.4 TASK SYNCHRONIZATION

DNOS provides synchronization on several functional levels. These levels correspond to the assumed commonality between the tasks requiring synchronization. The synchronization tools involved are interprocess communication (IPC) messages, semaphores, locks, and events. IPC is discussed in the section on I/O.

9.4.1 Semaphores.

Semaphores enable two tasks to exchange timing signals. A semaphore is implemented as an integer variable and a queue of waiting tasks. The integer variable indicates the number of unconsumed timing signals. If no signals are present, the integer indicates the number of tasks waiting for a timing signal.

Semaphore operations are provided on job-local variables via SVC >3D. The subopcodes in this SVC have the following meanings:

Subopcode 0 : SIGNAL
The value of the semaphore specified by byte 3 of the SVC call block is incremented. The oldest task queued on the semaphore queue is activated.

Subopcode 1 : WAIT
The value of the semaphore specified by byte 3 of the SVC call block is decremented. If the resulting semaphore value is negative, the task is suspended and queued to the specified semaphore.

Subopcode 2 : TEST
The value of the semaphore specified by byte 3 of the SVC call block is returned in bytes 4 and 5 of the SVC block.

Subopcode 3 : INITIALIZE
The semaphore specified by byte 3 of the SVC call block is initialized to the value specified in bytes 4 and 5 of the SVC block. If any tasks are queued waiting for this

semaphore, the action taken depends on the new value of the semaphore as follows:

- * If the new value is greater than or equal to 0, activate all suspended tasks.
- * Given that n is (new value - old value), if the new value is less than 0 and n is greater than 0, activate n tasks, starting with the oldest queued task.

Subopcode 4 : MODIFY

The semaphore specified by byte 3 is set to the sum of its old value and the two's complement (negative) value furnished in bytes 4 and 5 of the SVC block. If any tasks are queued waiting for the semaphore, the action taken depends on the new value of the semaphore, as described in the initialize operation. The modify operation combines the test and initialize operations so that correct results are obtained, even if other tasks are using that semaphore.

Semaphore values are represented as signed integers ranging from the negative value of -128 to a positive value of 127. A positive value indicates the number of signals sent but not received. A negative value represents the number of receivers waiting for signals unless the semaphore has been negative since the last time it was changed in a negative direction to a negative value by an initialize or modify operation.

9.4.2 Locks.

The synchronization tool available to tasks that share the same task address space is the lock. Locks enable tasks to implement mutual exclusion on critical sections of their code or data. Locks are represented as boolean data items, which indicate the state of the lock.

Locks can be implemented in assembly language by using the ABS and SETO instructions on a data variable. If tasks spend relatively little time executing in locked regions, the following code will achieve the desired mutual exclusion:

```
INITLK SETO LOCK        initialize the lock
.
.
AGAIN  ABS  LOCK        test the lock
       JLT  GOTLOK       * got the lock, use it
       SVC  TDELAY       * no, delay and retry
       JMP  AGAIN        *
GOTLOK . .
.
.
       SETO LOCK        free the lock
```

The higher-level synchronization primitives (SVC semaphores and IPC messages) are available to tasks at this level that do not meet the general assumptions about locks.

9.4.3 Event Synchronization.

To improve throughput, DNOS allows the execution of some SVCs in parallel with the task execution. The Initiate Event SVC (>41) provides this concurrency. It also eliminates polling in those situations where polling might be used because concurrency is unavailable. A set of 32 event flags is maintained in each TSB, showing which of the allowed 32 events is currently initiated or completed for the task.

The Initiate Event SVC points to an SVC to be initiated. An event number is generated by DNOS to identify this event. Event numbers range from 0 through decimal 31. In the current release of DNOS, I/O SVCs and semaphore SVCs can be initiated. If the operating system permits the specified SVC to be initiated, control is returned to the task after that request block has been buffered. If not, an error is returned to the user. The user must exercise caution, since the operating system may return information to the initiated SVC block at any time.

The Wait for Event SVC (>42) allows a task to wait for any of a set of events to occur. This SVC waits until one of several events has completed or until the maximum wait time is exceeded. The events to be waited for are specified by an event mask. The leftmost bit of the first word of the event mask corresponds to event 0. If this bit is set in the mask, the task is activated when event 0 is completed. The event flags returned to the call block indicate which (one or more) of the 32 events have completed.

The Post Event SVC (>4F) permits the user to post any event in any task in any job but the system job. This means that any Wait For Event may be aborted before its event is completed or its wait time has expired. The Post Event SVC should not be used to abort a wait for a valid initiated event; it should be used to provide a cross-job synchronization mechanism. If task A in job ONE executes a Wait For Event with a large time delay without initiating an event, then it will delay until either its time delay expires or it is posted, either from job ONE or from another job. This provides a method to deactivate and reactivate user-written queue server tasks, across job boundaries. To facilitate this operation, issuing a Wait For Event with a maximum time delay of -1 (>FFFF) is special cased to cause a virtually infinite maximum delay time.

SECTION 10

I/O SUBSYSTEM

10.1 OVERVIEW

The I/O subsystem moves data between any combination of logical and physical I/O resources and programs (tasks) that process the data. The logical resources are the files located on disk or magnetic tape and the channels between programs. The physical resources are the devices attached to the computer.

An I/O request enters the I/O system via the SVC interface. This interface provides resource-independent, resource-specific, and utility paths through a single SVC opcode, SVC 00. Most I/O is achieved via the resource-independent call because programmers usually want only to obtain data, process that data, and output the processed data without knowledge of special features of the I/O resource.

However, some special-purpose programs require knowledge of the I/O resource. They must use specific techniques and formats to obtain, process, and output data. These programs use resource-dependent I/O.

The utility path allows for dynamic management of resources without intervention from outside the computer. Actions such as reserving a resource, specifying access privileges, and releasing access are performed via the utility path.

The general form of the I/O SVC request block (IRB) is shown in the section on data structure pictures. The basic block is 12 bytes long, while the full IRB for complex requests is considerably longer.

An I/O request enters the I/O subsystem from RPROOT, the SVC decoder. The I/O system screens out the utility requests via the subopcode and passes them to the I/O Utility (IOU). The I/O system then finds the request routing information for those that are not utility requests. The routing information provides for checking on the operations allowed to the requester. A copy of the request call block is made in the STA so that the requester's memory space may be free for other tasks while the request is being processed.

The routing information is used to move the request to the correct resource handler. Channel requests are handed to the IPC processor. File requests are given to the file management (FM) processor. Device requests are handed to the device manager. The device manager is responsible for setting up the data buffer.

The request data buffer is moved to the buffer table area (BTA) if the destination resource transfers data relatively slowly. This copy of the request data buffer is made so that the task memory space may be released while the request is being processed. Resources that move data quickly need not have the data buffer copied; they access the data directly from the requester task memory space. The device manager passes the buffered request copies to the physical device handler, the DSR, for processing.

The DSR moves the data between computer memory and the physical device. This transfer usually occurs at the maximum rate of the device. During this transfer, the scheduler selects other programs to execute.

When the transfer has completed, the hardware causes a device interrupt to signal the DSR, indicating that the request has completed. The DSR sets up conditions such that the next time the scheduler selects a program to execute, it finds that the request has finished processing. The scheduler then activates the requesting task. Also, any request waiting to be processed by that DSR is passed to the DSR.

When a program is activated by the scheduler, a check is made before the program is allowed to execute to see if any buffered SVC requests are to be returned to the program. For I/O SVCs, status information and buffered data are returned.

10.2 DEVICE I/O DATA STRUCTURES

Data structures used for handling device I/O are of two types. One set describes the devices and is built by the system generation utility. The other type of structure is built by the I/O and I/O Utility subsystems when requests are made to use devices. The following structures are built during system generation:

* Physical device table (PDT) - Memory-resident data structure, one built for each device defined for the system. Contains information about the device name, characteristics, and workspace for the device service routine.

* Alternate PDT - A short version of a PDT, built for subdevices of a device. An example is the cassette unit of an ASR terminal. The DSFAID flag in the field PDTDSF identifies the PDT as an alternate. The field PDTDIB

points to the master PDT, that is, the PDT for which this represents a subdevice. The byte PDTTYP is a binary indicator of the subdevice. There can be no more than 256 subdevices per master PDT. Note that these structures must be carefully avoided by some processors; power-up must bypass all alternate PDTs, for example, and abort processing must also avoid them.

* Disk PDT extension (DPD) - Structure appended to the PDT for a disk device. Used as a work area by the device service routine and the Disk Manager.

* Keyboard status block (KSB) - Structure appended to the PDT for a device with a keyboard. Used as a workspace by the device service routine when handling the keyboard.

* Line printer PDT extension (LPD) - Structure appended to the PDT for a line printer device. Carries flags and pointers for use by the device service routine.

* Magnetic tape PDT extension (MTX) - Structure appended to the PDT for a mag tape device. Carries flags and counters for use by the DSR.

* Extension for a terminal with a keyboard (XTK) - Structure appended to the KSB for a device with a keyboard.

* Extension for a 940 or 931 terminal - Structures appended to the KSB for a 940 or 931 terminal. These are described in the paragraph on asynchronous DSRs.

The following structures are built when a request is issued to a device:

* Logical device table (LDT) - Built by the I/O utility to carry the logical unit number to be used for requests, characteristics of the device, and the current processing state.

* I/O request block (IRB) - Built by the I/O preprocessor as a buffered copy of the I/O SVC issued by the requester.

* Buffered request overhead (BRO) - Built by the request processing root and by the I/O subsystem to describe the originator of the IRB and the current state of the request, this is appended to the front of the IRB.

10.3 DEVICE I/O HANDLING

Figure 10-1 and Figure 10-2 show the flow of control through device handling. Figure 10-1 is an overall view, while Figure 10-2 details the entrance to device processing. The figures show the major I/O modules involved, as well as support routines from the nucleus and SVC request processing systems.

```
                          +-----------+
       +------------(XOP)-|REQUESTER  |<----------------------
       |                  |   TASK    |                     |
       V                  +-----------+                     |
  +-----------+                                             |
  |  RPROOT   |                                             |
  +-----------+                                             |
       |            +------->----->----------------         |
       V            |          |                  |         |
  +-----------+     |          V        ->--------|         |
  |  IOPREP   |     |    +-----------+  |          V         |
  +-----------+     |    |  NFSRTN   |  |   +-----------+   |
       |            |    +-----------+  |   |  NFTRTN   |   |
       V            |          |        |   +-----------+   |
  +-----------+     |          V        |         |         |
  |  IODEVR   |     |    +-----------+  |         V----->----
  +-----------+     |    |  NFSCHD   |  |   +-----------+
       |            |    +-----------+  |   |  RPDQUE   |
       .--------->---          |        |   +-----------+
     ...                     ...---->----        ...
```

Figure 10-1 Overview of Device I/O Handling

10.3.1 Details of I/O System Routines.

When the requester task issues an I/O SVC, control is passed to the SVC decoder, RPROOT. After determining that request is for the I/O system, RPROOT passes it directly to the I/O request preparation routine, IOPREP.

IOPREP functions as a preprocessor that is a uniform entrance into the I/O system and prepares the I/O request for the destination resource. If any error is detected by IOPREP, the error bit is set in the user call block flags, and IOPREP exits to RPRTNE in RPROOT. RPRTNE returns the error byte to the user call block and checks flags for initiated events.

```
                      +----------+
     +----------(XOP)-|REQUESTER |
     |                |   TASK   |
     V                +----------+
+----------+
|  RPROOT  |
+----------+
    (B)
     |
     V
+----------+
|  IOPREP  |
+----------+
    (B)
     |
     V
+----------+
|  IOCHKX  |
+----------+
    (B)
     |---------------+---------------+---------------+
     V               V               V               V
+----------+    +----------+    +----------+    +----------+
|  IODEVR  |    |  IUPREP  |    |  IPCPRE  |    |  FMPREP  |
+----------+    +----------+    +----------+    +----------+
   (BL)
     |--(B)----------------+----------------------+
     V                     V                      V
+----------+          +----------+           +----------+
|  IOPEL   |          |  NFSRTN  |           |  NFTRTN  |
+----------+          +----------+           +----------+
   (BL)
     |---------------------+----------------------+
     V                     V                      V
+----------+          +----------+           +----------+
|  IODBGN  |          |  LGDEV   |           |  NFEOBR  |
+----------+          +----------+           +----------+
  (BLWP)
     |
     V
+----------+
|   DSR    |
+----------+
   (BL)
     |---------------------+
     V                     V
+----------+          +----------+
|  IONRCD  |     OTHER DSR SUPPORT ROUTINES
+----------+          +----------+
```

Figure 10-2 Beginning Device Request Processing

IOPREP passes the request and control to IUPREP if the request is a utility request. Otherwise it builds a copy of the request in the STA static buffer, SYSBUF, including buffered request overhead (BRO) and the entire call block. IOPREP then calls IOFLDT to locate the LDT. The LDT contains information about the destination resource as a logical unit number (LUNO). If the resource pointer in the LDT is zero, the request is for the dummy device (DUMY); consequently, the request is simply returned as complete via RPRTNE.

For a device other than DUMY, the LDT is examined to see if the device has been opened, that is, some task has issued an I/O SVC with the Open subopcode. If the device is open, the LDT carries the TSB and JSB addresses of the task that opened it. The task attempting to use the resource must be the task that opened the LUNO.

If the LUNO is open to the requester task, various subopcodes in the I/O requests are treated differently. The Modify Access Privilege subopcode is treated as an Open. Otherwise, control is transferred around the open process. Read Characteristics subopcode requests are allowed to bypass the requirement that the LUNO be open.

If the LUNO is not open, the subopcode is checked to see that it is an Open. The open process checks for a Resource Privilege Block (RPB) to see if the Open is allowed. If it is, the LDT is opened and the access privileges are placed into the LDT and RPB. Requests of all subopcodes are channeled through the next part of IOPREP, which tests again to determine if the request is allowed with the current access privileges. If the request is legal, control is passed to IOCHKX for further processing.

IOCHKX buffers the remaining portion of the request into SYSBUF according to the device type specified in the LDT. The STA is used since it is available to devices and file management when the JCA is not in memory. The data buffer is not allocated or buffered at this time. Using more information from the LDT, control transfers to the IPC processor, IPCPR2; to the file management processor, FMPREP; or to the device processor, IODEVR.

IODEVR functions as a uniform I/O entrance to device resources. It has an alternate entry point (IODDIO) for direct device I/O of system tasks that must bypass checks on general requests. The primary entry point determines if a buffer should be allocated from the BTA and if data should be buffered. After these decisions are made, the paths from the two entry points come together.

IODEVR now checks to see if the device in use is represented by an alternate PDT, that is, is a subdevice of some master device. If so, the flag BRFAPI is set in the field BROOF2 of the buffered request overhead of the I/O request block. The binary ID of the device is copied from the alternate PDT to the field BROAID. The master PDT pointer is retrieved from PDTDIB of the alternate PDT and now used as the PDT pointer for processing.

The request is then inserted on the PDT waiting queue, PDTWQ, and control is given to the PDT end-of-record logic routine, IOPEL. When control returns from IOPEL, the request is checked to see if it is complete in IORTN. If so, control passes to the nucleus return routine, NFTRTN. If the request is not complete and it is not an initiate mode request, the task state is set to suspended for I/O, and control is given to the nucleus suspend routine, NFSRTN. If the request is an initiate mode request, control is given to NFTRTN.

IOPEL functions as a device control module outside of hardware interrupts, setting up requests to devices and returning requests to tasks. The loop for processing begins by calling a system log routine, LGDEV, to log any errors stored in the PDT. Requests are set up for the device if the PDT saved request block address, PDTSRB, is zero and PDTWQ is not zero. Before any processing, IOPEL sets the hardware interrupt level in the status register to prevent interrupts. The first request is removed from PDTWQ, the device map file is changed to map in the request, the data buffer address in the BRB is adjusted, and control is given to the device begin routine, IODBGN.

When control returns from IODBGN or if PDTSRB is nonzero, the PDT spent request queue, PDTSRQ, is examined. If PDTSRQ is nonzero, a request is removed from it, and NFEOBR is called to insert the request on the TSBEOR queue. This then loops back to the logging process. If the device is busy or there are no more waiting requests and no more completed requests, control returns to the calling routine.

IODBGN is an interface routine that changes the map file from the current state to a state in which the DSR is mapped with its data buffer. IODBGN must be in the first of the three segments of map file 0 in order to perform this function. After the new map file has been set up, the DSR is entered at the request entry point (one of several entry points). Alternate entry points in IODBGN correspond to some of the other entry points in the DSR. These alternate entry points are IODREE for system interrupt entry, IODABT for request abort, and IODTO for time-out, IODPDS for priority scheduling, and IODPU for power-up. Before giving control to the DSR, IODBGN saves the address of the PDT workspace for power failure.

10.3.2 I/O Processing by the DSR.

A DSR is the request processor for a physical resource. The first five instructions beginning at relative location 0 of the DSR must be branch instructions. These branch instructions correspond to five alternate entry points in the DSR. A sixth branch instruction must be included if the DSR uses priority DSR scheduling. The branch instructions correspond to the following alternate entry points.

* Hardware interrupt, the routine that handles interrupts from the device

* System interrupt, the routine that handles the request for the system to reenter at approximately 50 milliseconds later.

* Power up, the routine that initializes the device.

* Request abort, the routine that handles the abort of a request that the DSR is processing

* Request time-out, the routine that processes the condition in which the device has not responded in a certain length of time.

* Initial request processing. If priority DSR scheduling is used then this must be a branch instruction to the routine that handles initial request processing. If priority DSR scheduling is not used, then initial request processing code begins here.

* Priority DSR scheduler (optional)

If priority DSR scheduling is used, the instruction following the initial request processing entry point is the routine for processing requests for priority DSR scheduling. Priority DSR scheduling is used by DSRs which need to be reentered but do not want to wait the 50 milliseconds for a system interrupt. This mechanism reenters the DSR after all interrupt processing to the system is complete but before the task scheduler initiates any task execution.

DSRs which use priority scheduling must link in the routine IOPDSQ. The DSR requests priority scheduling by issuing a BLWP @IOPDSQ instruction. The routine IOPDSQ will queue the PDT to the priority scheduling queue. NFSCHD and NFTRTN check for PDTs on this queue and reenter the DSR at the earliest opportunity. This is intended for use only by high priority interrupt processing. Using this mechanism arbitrarily may interfere with other devices that use this entry point.

When a hardware interrupt occurs, the interrupt vector tables (initialized during sysgen) are used to transfer to the appropriate interrupt decoder. There are four decoders provided with DNOS, each serving a class of devices:

* A single device at a unique interrupt level

* Multiple devices at a single interrupt level

* An expansion chassis at a single interrupt level with multiple devices, each at a unique interrupt level within the chassis or multiple devices sharing a unique interrupt level within the chassis.

* Single device or multiple devices on a multiplexed device controller

Each interrupt decoder goes through the following steps to process the interrupt:

1. Save the current system map file pointer (accessed via CURMAP).

2. Set CURMAP to the DSR map file pointer for the appropriate DSR.

3. Load the map file using CURMAP.

4. Enter the DSR at the hardware interrupt entry.

5. When the DSR completes, restore CURMAP to point to the previous system map file.

6. Load the map file using CURMAP.

7. Exit via NFTRTN.

The text of the interrupt decoder can be found in the section on writing a DSR in the <u>DNOS Systems Programmer's Guide</u>.

Whether handling hardware interrupts or entering the DSR at other points, the DSR uses the PDT for the device as a reference point. The section on data structure pictures includes details on the PDT.

A queue header in the PDT allows the DSR to accept multiple requests for the device and to call the DSR end-of-record routine, ENDRCD, as many times as necessary to dispose of completed requests. (ENDRCD is one of several routines in the module IONRCD.) To handle the multiple requests, the DSR must remove the request from PDTSRB (clearing PDTSRB) and insert the request on the PDT hidden request queue, PDTHRQ. By clearing

PDTSRB, the DSR appears to be not busy. PDTHRQ is used as an internal queue anchor for the DSR; the operating system can use PDTHRQ to abort requests or to allow the task to wait for requests.

ENDRCD is the first step in returning the request to the task. Not much can be done at this level because of the time spent with hardware interrupts masked to the interrupt level of the device. ENDRCD expects PDTSRB to contain the address of the request that has just completed. If PDTSRB is zero, ENDRCD returns to the DSR. If PDTSRB is nonzero, ENDRCD removes the request from PDTSRB, clears PDTSRB, and inserts the request at the end of PDTSRQ. If the PDT end-of-record queue (PDTERQ) is zero, the PDT is inserted at the end of the list of PDTs that need end-of-record processing. This list is anchored by EORNKR, found in NFPTR, and has PDTs queued via their PDTERQ fields. The priority of the executing task is compared with the priority for the requesting task. If the request priority is higher, the global forced reschedule flag, RESCHD (found in NFDATA), is set to preempt the running task. Control then returns to the DSR.

Figure 10-3 shows timing, system interrupt, hardware interrupt, and end-of-record support for DSRs. The figure highlights only the main modules or routines.

The task scheduler, NFSCHD, interacts with the I/O system to handle system interrupt and end-of-record functions. When a system time unit (50 milliseconds) has elapsed, NFSCHD calls the device timer routine, IODTMR. IODTMR traverses the PDT list, examining it for flags set to reenter a DSR and to wait for request time-out. When the reenter me flag is on, it is turned off and IODREE is called. If the PDT is busy, the time-out flag is on, and the PDT times out, an error code for device time-out is placed in the active request and IODTO is called. (IODREE and IODTO are alternate entry points to module IODBGN.)

NFSCHD determines that device end-of-record processing is required by finding a nonzero value in the global queue anchor, EORNKR. When this occurs, the end-of-request routine for PDTs, IOEOR, is called. IOEOR removes the first PDT on EORNKR and then calls IOPEL to process the end-of-record. IOEOR removes each PDT from the list until it is empty. IOEOR then returns to the scheduler.

```
                       +----------+
                       |  NFSCHD  |
                       +----------+
                           (BL)
                            |
     -----------------------+
     |                      |
     V                      V
+----------+           +----------+
|  IODTMR  |           |  IOEOR   |
+----------+           +----------+
    (BL)                   (BL)
     |                      |
     |                      V
     |                 +----------+
     |                 |  IOPEL   |
     |                 +----------+
     |                     (BL)
     |                      |
     | +--------------------+-----------------------+
     V V                    V                       V
+----------+           +----------+            +----------+
|  IODBGN  |           |  LGDEV   |            |  NFEOBR  |
+----------+           +----------+            +----------+
   (BLWP)
     |           (BLWP) +----------+
     V------<-----------| hardware |
+----------+            | interrupt|
|   DSR    |            |  decoder |
+----------+            +----------+
    (BL)
     |----------------------+
     V                      V
+----------+     +--------------------------+
|  IONRCD  |     |OTHER DSR SUPPORT ROUTINES|
+----------+     +--------------------------+
```

Figure 10-3 DSR Control Paths

10.3.3 Returning Information to the Requester.

Figure 10-4 shows how information is returned to the requester task. After a task is scheduled and prior to execution, NFTRTN is called. (NFTRTN is also the exit point for nonsuspending SVC processors.) Before NFTRTN returns control to the requester task, the TSBEOR field of its TSB is examined for a nonzero value. When TSBEOR is zero, control drops through for the other checks. Otherwise, RPDQUE is called.

NFSRTN is similar in nature to NFTRTN except that it is the exit point for SVCs that suspend task scheduling. Before suspending the task, NFTRTN examines TSBEOR for a nonzero value. When it is zero, control drops through and the task is suspended. Otherwise, RPDQUE is called.

RPDQUE removes the first entry from TSBEOR. RPDQUE examines the SVC code in the BRB, unbuffers the error code, and calls the SVC post processor if one exists. The post processor for I/O is IOPOST. When control returns from IOPOST, RPDQUE releases the BRB, checks for another entry on the queue and processes any. When no entries remain, it returns to its caller.

IOPOST examines the subopcode to determine if it is an I/O utility opcode. If so, parameter buffers are released and other information is unbuffered to the task. If the request was made through IODDIO in IODEVR, the LDT address is zero and requests are not unbuffered. Otherwise, the system flags are unbuffered. For relative record files, the record number is returned to the task. For VDT requests, the extended user information is unbuffered. The SVC subopcode is then used to index into a table to unbuffer Open, Read, and Write information.

For Open subopcodes, the resource type is returned to the task. If an error code is in the BRB for an Open or a Close, both the LDT and the RPB are closed. For a Read subopcode, the data buffer is moved to the task data buffer if the buffer beet address in the BRO, BROBBA, is a nonzero value. If BROBBA is zero, the buffer has already been moved by some other processor. Control then passes back to the caller, RPDQUE.

```
                         +-----------+
                         |REQUESTER  |<-------------------------+
                         |   TASK    |                          |
                         +-----------+                          |
                                                                |
                   +-----------+           +-----------+        |
                   |           V           |           V        |
 +-----------+     |     +-----------+     |      +-----------+ |
 |  NFSRTN   |     |     |  NFSCHD   |     |      |  NFTRTN   | |
 +-----------+     |     +-----------+     |      +-----------+ |
     (BL)          |         (B)           |          (BL)      |
      |-(B)--------+          +------------+           |        |
      |                                                |--(RTWP)+
      | +----------------------------------------------+
      V V
 +-----------+
 |  RPDQUE   |
 +-----------+
     (BL)
      |-----------------------+
      V                       V
 +-----------+    +-------------------------+
 |  IOPOST   |    |OTHER SVC POST PROCESSORS|
 +-----------+    +-------------------------+
```

Figure 10-4 Returning Information to the Requester

10.3.4 Bidding a Task from a DSR.

DNOS provides a means to bid a task from a DSR. A two character sequence must be entered to initiate the task bid. The first character entered is the arming character, Attention. The second character entered is used to identify the task to be bid. For example, the sequence Attention ! can be used to bid SCI. In addition to the task bid sequences, DNOS processes the following character pairs as indicated:

Attention Attention - halt current output to screen,
 resume output
Attention Return - abort I/O to the screen
Attention Control-X - break - terminate current task
Attention N - bid the network logon task

These can be redefined or more character sequences can be defined by the user. For each task to bid, the user must supply a Command Definition Entry (CDE). The CDE is associated with the type of device at which the bid can be made. During IPL, a file of CDE tables is initialized for each type of device that might be used for task bidding. This file is assumed to be in VOL.S$CDT.xxx, where VOL is a synonym for the disk volume being used as a data disk and xxx is the name of the generated system.

The format of a CDE is shown in the section on data structure pictures. Each CDE includes a task ID for a logon task to be bid, as well as a task ID for the task to be bid by the logon task, flags, and parameters for the task to be bid by the logon task. The logon task is either the task supplied with DNOS or some user-written substitute. The supplied logon task solicits user ID and passcode, verifies their accuracy, and bids the task specified in byte 3 of the CDE. (More detail on the logon task can be found in the section on system tasks.)

When a keystroke defined by a CDE is used at a terminal, the DSR makes several entries to system data structures. If no other task is currently awaiting bid for the terminal in question, the PDTCHR field of the terminal PDT is set to the character entered. The PDT is linked to the global list of PDTs with pending bids, using the PDTBQ field as a link field. The global list is anchored at BIDREQ, located in NFPTR.

When the scheduler is scanning lists to find an activity to begin, it examines the BIDREQ anchor to see if any PDT needs a task bid. If the anchor is nonzero, the scheduler bids the system task IOTBID to serve the queue of requests.

IOTBID takes the first request from the queue and examines it for validity. It first ensures that the terminal is on-line and available for use.

IOTBID then issues a Bid Task from a DSR (>C7) subopcode of the I/O SVC to bid the appropriate task as defined by the CDE. Refer to the section on the device I/O utility (DIOU) for more details of that bid process.

10.3.5 Handling Large I/O Buffers.

The use of full-duplex operations for communication requires a strategy to handle many large data buffers. DNOS uses the BTA for large buffers rather than allocating them from STA. Figure 10-5 shows in general how buffers are mapped with I/O processing.

The BTA was designed to accommodate transient buffers; therefore, the BTA dynamically expands and contracts. During sysgen, static allocation limits are set. By using the Static Buffer Forced Roll SVC (>4A), the system can interrogate, increase, or decrease the size of the BTA. The BTA immediately precedes available user memory; when increased, the BTA causes user memory to decrease. This may cause forced swap of user segments that are occupying the requested area. SVC >4A is detailed in the section on special SVC support.

Since the buffers are not in STA, it is possible to dynamically get and release BTA buffers from a DSR. Subroutines are provided for the DSR to get BTA (IOGBLK) or release BTA (IORBLK). To use IOGBLK, BL to the subroutine with workspace register 10 (R10) containing the size of the buffer (bytes). On returning from the subroutine, workspace register 0 contains the status code for the request. It will be 0 if no errors occurred while getting BTA. R10 will contain the beet address of the allocated memory. The value returned in R10 should be stored in the buffered beet address field (BROBBA) of the BRB that will be receiving this buffer. The I/O system uses BROBBA as a beet address within the BTA to release the buffer. The first usable address relative to the start of the allocated BTA buffer is the value of BBAOFF found in the CSEG NFWORD.

The BTA can be addressed locally by the DSR if the BTA is mapped in. Before mapping in the BTA, it is necessary to map out the buffer currently mapped in (if one is present and if it is different from the new one). This is accomplished by a call to the subroutine IOMPOT, which uses R1 as a pointer to the BRB containing the pointer to the buffer to be mapped out. Mapping in the new buffer is achieved by pointing R1 and PDTSRB to the desired BRB and by placing the value of BBAOFF in the IRBDBA field of the BRB. Then the DSR calls the subroutine IOMPIN to map in the BTA. This causes the IRBDBA field to contain a buffer address within the logical address space of the DSR.

To release BTA, the DSR calls the subroutine IORBLK, with R10 containing the beet address in the BTA of the buffer to release. On returning from the subroutine, R0 contains the status code for the request. R0 will be zero if no error was detected. After a buffer in the BTA is released, BROBBA in the corresponding BRB must be set to zero.

Control of buffering for a DSR is specified by information contained in the PDT of each device. PDTDSF contains the two flags, DSFBI and DSFBO. DSFBI controls the input. When it is 1, a buffer is allocated in the BTA for a read request. When it is 0, a buffer is not allocated for read requests. DSFBO controls the output. When it is 1, a buffer is allocated in the BTA for a write request, and the output data is copied to the buffer from the task address space. When DSFBO is 0, a buffer is not allocated and not copied for write requests.

```
                       Physical memory

+-----------------------------------------------------------+
|DNOS|Memory-Resident|   |Memory-Resident| Buffer |  User   |
|Root|   DNOS Code   |DSR|   DNOS Code   |  Table |  Task   |
|    |               |   |               |  Area  | Memory  |
+-|--------------------|-------------------|---------|------+
  |                    |                   |         |
  |                    |  +----------------|         |
  |                    |  |  +-----------------------|
  |                    +--|--|-------+
  |                       |  |       |
  +-----------------+     |  |       |
                 +--|-----|--|-------|---+
                 | DNOS |  Data   |      |
                 | Root | Buffer  | DSR  |
                 +-----------------------+
                 0               6000

                     DSR Logical Memory
```

Figure 10-5 Device I/O Buffering

10.3.6 Converting a DX10 DSR for DNOS.

Because of different internal operating system structures and because of some added functions, DNOS DSRs are slightly different from their DX10 counterparts. A user who has his own DX10 DSR must change his DSR to meet DNOS standards.

Before making any code changes, the user should study the relevant data structure templates used in DNOS: the PDT, IRB, BRO, and any other templates whose counterparts were used in the DX10 DSR.

Within the DSR code itself, the set of definitions (DEFs) and references (REFs) must be changed. The DNOS I/O system uses no DEFs supplied by the DSR. The only DEFs that must be supplied are those required by modules used by the user's DSR. Any other DEFs may be deleted.

Several DX10 REFs are not used by the DNOS DSR. Delete the REFs to the subroutines SETWPS, BZYCHK, and MAPCHK. Replacements for these references are discussed in the following paragraphs. The REF for KEYFUN should be replaced by IOFCDT, and the template for NFPTR, DSC.TEMPLATE.COMMON.NFPTR, should be copied into the DSR. The REF for BRCALL or JMCALL should be replaced by BRSTAT. REFs to byte and/or word constants should be deleted and replaced by copying in the appropriate template:

BYTE00 - BYTE0F	DSC.TEMPLATE.COMMON.NFER00
BYTE10 - BYTE1F	DSC.TEMPLATE.COMMON.NFER10
BYTE20 - BYTE2F	DSC.TEMPLATE.COMMON.NFER20
BYTE30 - BYTE3F	DSC.TEMPLATE.COMMON.NFER30
BYTE40 - BYTE4F	DSC.TEMPLATE.COMMON.NFER40
BYTE50 - BYTE5F	DSC.TEMPLATE.COMMON.NFER50
BYTE60 - BYTE6F	DSC.TEMPLATE.COMMON.NFER60
BYTE70 - BYTE7F	DSC.TEMPLATE.COMMON.NFER70
BYTE80 - BYTE8F	DSC.TEMPLATE.COMMON.NFER80
BYTE90 - BYTE9F	DSC.TEMPLATE.COMMON.NFER90
BYTEA0 - BYTEAF	DSC.TEMPLATE.COMMON.NFERA0
BYTEB0 - BYTEBF	DSC.TEMPLATE.COMMON.NFERB0
BYTEC0 - BYTECF	DSC.TEMPLATE.COMMON.NFERC0
BYTED0 - BYTEDF	DSC.TEMPLATE.COMMON.NFERD0
BYTEE0 - BYTEEF	DSC.TEMPLATE.COMMON.NFERE0
BYTEF0 - BYTEFF	DSC.TEMPLATE.COMMON.NFERF0
WD0001 - WD8000	DSC.TEMPLATE.COMMON.NFWORD

To allow for the addition of new devices at IPL and for the reinstallation of a new, modified, or corrected DSR without linking the entire system, the first addresses of the DSR must be the following instructions:

```
B    @Hardware interrupt entry address
B    @System reenter me entry address
B    @Power up entry address
B    @Abort entry address
B    @Time-out entry address
B..  @Request entry address
B    @Priority DSR scheduler
```

No data or subroutine code can precede these instructions. They replace the following DX10 entry points:

```
DATA power-up entry address
DATA abort entry address
...   DSR executable code for request entry
```

In DX10, the following is the first DSR executable code:

```
LIMI 0
BL   @SETWPS
```

This code is not required and should be deleted for DNOS because the I/O system performs this function prior to entering the DSR.

Two differences in the data structures affect code in the DSR. They involve the pointers in R1 and R4 of the PDT. R1 is the pointer to the BRB. The BRB, a concatenation of the BRO and the IRB, is called the UCB in DX10. The relevant change in DNOS is the position to which R1 points in the BRB. In DX10, R1 pointed to the word containing the subopcode and LUNO of the BRB. In DNOS, R1 points to the word containing the SVC code and error byte. PDTSRB points to the same place. The DSR code must be changed to reflect the new pointer. Any references to the DX10 structures named PRB and UCB must be changed to the equivalent references to the DNOS structure named IRB.

R4 points to the device information block (DIB) of the PDT. (The device information block is the PDT and any PDT extensions.) In DX10, R4 pointed to the word beyond the end of the PDT. In DNOS, R4 points to the first word of the PDT, the PDT link word PDTPDT. The DSR must be changed to reflect the new pointer location. References to DX10 PDT offsets must be changed to equivalent references to PDT template offsets. PDT labels should be used rather than hard-coded offset values.

The subopcode processor, BRCALL or JMCALL, has been enhanced to collect information for on-line diagnostics. BRCALL can still be called, but note that R10 will be modified. The replacement subroutine call is to BRSTAT. It uses R10 as a pointer to a byte table that contains relative offsets into the PDT. If R10 is zero, it is not used as a pointer. The table is built with entries for each subopcode. The on-line diagnostics code counts types of requests for physical I/O. The entries in the byte table are chosen from 0, PDTRC, PDTWC, or PDTMC; these correspond to the null request counter, the read request counter, the write request counter, and the miscellaneous request counter, respectively. Build the diagnostics table appropriately for the physical device if this function is chosen. The table for BRSTAT is the same as for BRCALL.

One of the subroutines for keyboard devices has a different name in DNOS and DX10. The subroutine KEYFUN in DX10 is named IOFCDT in DNOS. IOFCDT performs the same function as KEYFUN; it also performs a new function, bidding a task from a DSR. (See the paragraphs describing bidding a task from a DSR for details.) All task bids in a DX10 DSR must be removed in the DNOS equivalent.

IOFCDT controls processing of all bids, including the bid of SCI at the terminal. (SCI may be bid with or without the logon task.) IOFCDT also processes the hard break sequence, bidding the IOBREAK task. The keys used to bid SCI and the hard break sequence are defined in the CDE for the terminal type during IPL.

For devices with no KSB associated with the PDT, a bid can be accomplished by direct queue manipulation in place of a call to IOFCDT. To place the entry on the queue, first examine the byte field PDTCHR. If the field is nonzero, a task is waiting to be bid and another entry is not allowed. If PDTCHR is zero, place the character in the PDTCHR field. Mask interrupts to level 2 and find the end of the queue whose anchor is BIDREQ (found in NFPTR). Link the PDT to the last one on the queue using the field PDTBQ. Clear PDTBQ in this PDT, and enable interrupts to the proper level.

The REFs for BZYCHK, SETWPS, and MAPCHK must be deleted. Replace the call to BZYCHK with code like the following:

```
MOV  @PDTSRB(R4),R7
JNE  device busy code
RTWP
```

The function previously performed by MAPCHK is now performed by the I/O system prior to entering the DSR. Therefore, any code that references MAPCHK must be removed from the DX10 DSR for use with DNOS.

In addition to the features already mentioned, some others are provided by the I/O system. The error code is placed into the BRB in power-up and abort situations. The error flag for the IRB is
set in the BRB whenever a nonzero value is detected in the error byte of the BRB.

For processing an end-of-record for a DSR, the subroutine ENDRCD has been enhanced to accommodate successive calls to perform multiple end-of-record requests. To take advantage of this feature, correctly set up the pointer PDTSRB before calling ENDRCD. PDTSRB is the pointer to the saved request block and must point to the SVC and error byte of the BRB. If PDTSRB is zero, nothing occurs in the subroutine ENDRCD.

For a DSR that must multiplex its input and output, a queue anchor for this purpose is included in the PDT. When a DSR wishes to receive a second request, it must appear to the I/O system 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 hidden request queue, PDTHRQ, using the link word BROBRO in the BRB. In this way, the I/O system can find a request being aborted and flag the error byte with a >10 error code. 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.

If the DSR multiplexes two or more requests at the same time, it must be careful when accessing a buffer. The buffer for only 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.

While it should cause no code changes, the size of the PDT in DNOS is larger than that in DX10.

Special problems that will cause some re-design of the DSR are the inaccessibility of the LDT and the TSB. Although pointers exist in the BRO portion of the BRB, the segments containing the structures may be swapped out of memory. No mechanism is provided to the DSR to place one of these structures into memory or to map the structure into the logical address space of the DSR.

Typical problems encountered while debugging the converted DSR are attempts to access flags contained in the PDT registers and improper use of the pointers in R1 and R4. Check the PDT flags used by the DSR to make certain that they exist and are referenced by label.

The problems associated with R1 and R4 usually result from using the same method of reference in DNOS as in DX10. Since the values in R1 and R4 are pointers to the start of a structure, referencing must be via the appropriate template fields as follows:

```
DX10..                        DNOS........
   MOV   *R1,...                MOV   @IRBSOC(R1),...
   MOV   *R4,...                MOV   @PDTSIZ(R4),...
   ABS   *R4                    ABS   @PDTSIZ(R4)
                     or
   MOV   ...,*R1                MOV   ...,@IRBSOC(R1)
   MOV   ...,*R4                MOV   ...,@PDTSIZ(R4)
```

After assembling the DSR, it must be linked with all of the required support subroutines. This must be done with each release of the operating system, not with each sysgen between releases. Figure 10-6 shows a typical link control stream used to link a DSR.

```
NOPAGE
ERROR
FORMAT COMPRESSED
PROCEDURE DUMROOT
DUMMY
INCLUDE     VOL.S$SGU$.DUMROOT
PHASE 0,DUMROOT
DUMMY
PHASE 1,DSRname,PROG >C000
INCLUDE     VOL.DSRobject pathname
INCLUDE     VOL.IOMGR.OBJECT.IONRCD
   (include any other support routines)
END
```

Figure 10-6 DSR Link Control Stream

The linked object should be placed in the file VOL.S$SGU$.DSRname. VOL is a synonym for the volume name of the data disk being used for sysgen.

Table 10-1 shows the modules required for the support subroutines and the registers altered by each of the subroutines.

Table 10-1 Location of Support Subroutines for DSRs

		Registers
Module Name	Subroutine	Changed
VOL.IOMGR.OBJECT.IOBMGT	IOMPIN	R0
	IOMPOT	R0
	IOGBLK	R0,R10
	IORBLK	R0
VOL.IOMGR.OBJECT.IOKB	IOFCDT	R6
	CMODE	R5,R7,R9,R10
	PUTEBF	--
	PUTCBF	--
	GETC	R9
	ASCCHK	R10
	ASCCK2	R10
VOL.IOMGR.OBJECT.IONRCD	BRSTAT	R0,R10
	BRCALL	R0,R10
	ENDRCD	R0,R10
VOL.IOMGR.OBJECT.IOTILN	GTADDR	R9,R10
	XFERM	R6,R9,R10
	TILERR	R8,R9

10.4 TELEPRINTER TERMINAL DSR

DNOS contains several hardcopy terminal-driver DSR's:

DSR	TERMINALS	FUNCTIONALITY
DSRTPD	703,707,743,745,763,765, 78X,820,825	Local, remote KSR
DSRKSR	733, 742, 782	Local, remote KSR

This section describes details about DSRTPD as a detailed example of a DNOS DSR.

Direct connection for teleprinter terminals is supported using the following cable combinations:

TERMINALS	CONTROLLERS			
			/10A 9902 PORT	S300 AUX 2 PORT
	TTY/EIA	COMMIF	CI402	CI422
743/745	0948968-0001	0946117-0001 +2263351-0001 +0983848-0001		
763/765	2265151-0001 +2263351-0001 +2200051-0001	0946117-0001 +2263351-0001 +2200051-0001		
78X/820	2262093-0001	0946117-0001 +2263351-0001 +2207634-0001 or0946117 +0993210-0001	2303077-0001	2230504
703/707			2303077-0001	2230504

Full-duplex modems compatible with Bell 103, 212a, 113, and Vadic VA3400 series are supported, with cable 2265151-0001 from the TTY/EIA board, cable 946117-0001 from the COMM board, cable 2303070-0001 from non S300 9902 ports, and cable 2532883-0001 from S300 9902 ports. Half-duplex operation is not available to these terminals through the TTY/EIA interface module. Auto-call support is provided through the Teleprinter Device Utilities of SCI.

DNOS has adopted a philosophy of generating a dedicated DSR for each kind of I/O device supported, and of limiting access to interface cards (within a given configuration) to one of these dedicated DSRs. DSRTPD is to some extent a departure from this approach, as it allows a number of different kinds of computer terminals to be serviced from one DSR and one piece of interface hardware at different times without re-generation of the operating system. SCI functions with DSRTPD and the DNOS sysgen processor comprehends the parameters and PDT structure required by DSRTPD.

10.4.1 DSRTPD Structures.

The teleprinter device family (designated KSR) is identified by sysgen to include a wide variety of teleprinter terminals. The teleprinter device type code returned by an open operation is >0001 for this device family. The resource type returned on an assign luno for the teleprinter device family is >0902.

10.4.2 PDT Structures.

The TPD PDT is structured as follows:

```
+-----------------------+
|                       |
| PDT (built by SYSGEN) |
|   non-interrupt WS    |<---+
|   DNOS flags          |    |
|xxxxxxxxxxxxxxxxxxxxxxx|    |
|                       |----+
| KSB (built by SYSGEN) |
|   interrupt WS        |----+
|   DNOS flags          |    |
|xxxxxxxxxxxxxxxxxxxxxxx|    |
|                       |    |
| DIB (built by SYSGEN) |    |
| working parameter set |    |
| scratch area          |    |
| default parameter set |    |
| error counters        |    |
|_______________________|    |
| KSBCBF                |<---+
| input character buffer|
| (built by SYSGEN)     |
|_______________________|
```

The Device Information Block (DIB) is a data structure appended to the PDT which contains information about the current status of the device as well as information about how it was configured during system generation. The DSRTPD DIB has the following structure:

```
                          '*' DENOTES FIELDS INITIALIZED BY SYSGEN

DIBACR DATA 0             *ACU CRU ADDRESS(>FFFF IF NONE)
DIBHWR BYTE 0             *INTERFACE TYPE
*                             1=COMM/IF
*                             2=FCCC
*                               3=BCAIM
*                               4=HSCC
*                               5=TTY/EIA
*                               6=9902
       BYTE 0                RESERVED
DIBRTO DATA 0               *READ TIMEOUT (IN 1/4 SECONDS)
DIBWTO DATA 0               *WRITE TIMEOUT(IN 1/4 SECONDS)
DIBDT1 DATA 0               *FIRST DIRECT TIMEOUT (IN 1/4 SEC)
DIBDT2 DATA 0               *SECOND DIRECT TIMEOUT (IN 1/4 SEC)
DIBGFL FLAGS 8              *SYSGEN FLAGS (SAME AS DIBTFL)
       FLAG   GFLECO            ECHO (1=NO ECHO)
       BITS   1                 UNUSED
       FLAG   GFLXPE            XMIT PARITY ENABLED(1=ENABLED)
       BITS   2                 XMIT PARITY TYPE
*                               00=EVEN
*                               01=ODD
*                               10=MARK
*                               11=SPACE
       FLAG   GFLRPE            RECEIVE PARITY ENABLED
       BITS 2                   RECEIVE PARITY TYPE
DIBSTF FLAGS 8               STATE FLAGS
       FLAG   STFONL            0=ONLINE
       FLAG   STFCIP            1=CONNECT IN PROGRESS
       FLAG   STFOPN            2=OPEN
       FLAG   STFDLE            3=DLE RECEIVED
       FLAG   STFHDX            4=HDUX LINE BELONGS TO REMOTE
       FLAG   STFRSD            5=RESEND FLAG
                                6=UNUSED
                                7-8=BIT DATA QUEUEING
DIBLNF FLAGS 8              *LINE FLAGS
       FLAG   LNFHDX           *HALF DUPLEX (1=HALF DUPLEX)
       FLAG   LNFSWT           *SWITCHED LINE (1=SWITCHED)
       FLAG   LNFRCL            REFUSE CALL
       FLAG   LNFADE            AUTO-DISCONNECT ENABLED
       FLAG   LNFDLE            DLE/EOT FOR DISCONNECT
       FLAG   LNFSCF            SCF READY/BUSY MONITOR
       FLAG   LNFEXC            FILE XFER EXCLUSIVE ACCESS
       FLAG   LNFHDL            HALF DUPLEX LTA ENABLE
DIBTFL FLAGS 8               TEMPORARY ACCESS FLAGS
       FLAG   TFLECO            ECHO (1=NO ECHO)
       BITS   1                 UNUSED
       FLAG   TFLXPE            XMIT PARITY ENABLED
       BITS   2                 UNUSED
       FLAG   TFLRPE            RECEIVE PARITY ENABLED
DIBSPD BYTE 0               *BAUD RATE (SPEED)
*                              -1=300 OR 1200 SELECTED
*                                   BY 212 MODEM
*                               0=110
*                               1=300
*                               2=600
*                               3=1200
*                               4=2400
*                               5=4800
*                            6=9600
DIBEOR BYTE 0              *END OF RECORD (=CR)
DIBEOF BYTE 0              *END OF MEDIUM (=EM)
DIBLTA BYTE 0              *LINE TURNAROUND (=EOT)
DIBSUB BYTE 0              *PARITY ERROR SUBSTITUTE (='?')
DIBDLA BYTE 0               CARRIAGE RETURN DELAY INTERVAL
DIBPCR DATA 0               PARITY CHECK ROUTINE ADDRESS
DIBPSR DATA 0               PARITY SET ROUTINE ADDRESS
DIBMXC DATA 0               MAXIMUM CHARACTERS BUFFERED
DIBTRM BYTE 0              *TERMINAL TYPE (=TYPE-700)
DIBLCR BYTE 0               LAST CHARACTER RECEIVED
DIBXFL FLAGS 16             SAVED EXTENDED FLAGS
DIBSVE BYTE 0               SAVED ERROR CODE FROM DSR
DIBGSP BYTE 0              *CURRENT SPEED (SAME AS DIBSPD)
DIBISR DATA 0               RESERVED
DIBGTO DATA 0              *GENNED TIMEOUT(IN 1/4 SECONDS)
DIBPEC DATA 0               NUMBER OF PARITY ERRORS
DIBLCC DATA 0               NUMBER OF LOST CHARACTERS
DIBSIZ EQU  $-DIBBGN        SIZE OF DIB
```

10.4.3 DSRTPD Functions.

The DSR has a power up entry point labelled PWRON. Powerup processing consists of copying default parameters to the DIB, setting the state of the interface module accordingly, and setting values for the EIA lines which are appropriate to the line mode:

1. For switched lines, all lines are forced low until Ring Indicate is sensed or until the modem's online signal is detected: Data Carrier Detect for full duplex, Data Set Ready for half-duplex. Because DCD is not present for the 9902 port on the /10A controller, switched line is not supported for that configuration.

2. For unswitched lines, Data Terminal Ready is asserted and the DSR looks for Data Set Ready before each character is transmitted.

The DSR has an abort I/O entry point labelled ABORT. Abort I/O processing consists of terminating any I/O in progress with the abort error code (>10) and returning the DSR to the idle state. Timeouts appear to DNOS to be disabled, but the DSR handles timeouts internally.

10.4.4 DSRTPD Details.

DSRTPD is built of five modules: DSRTPD, DSCOMISR, DSTTYISR, DSISR402, and DSTPDCOM. The DSR uses a vector table to access various hardware-related functions.

DSRTPD

This module is the request processor interface. Its code is completely hardware independent. When a hardware-dependent function needs to be performed, it goes through a vector table and enters the appropriate hardware-dependent module. This module uses the PDT workspace exclusively. It maintains a set of state tables that are used to control interrupt-driven functions.

DSCOMISR

The COMM board driver module is named DSCOMISR. It contains code that is executed from both the PDT and KSB workspaces.

DSTTYISR

The TTY/EIA driver is called DSTTYISR.

DSISR402

All 9902 controllers (CI402, CI421, CI422, 110A 9902 port) use this hardware driver. It contains code that is executed from both PDT and KSB workspaces.

DSTPDCOM

This module performs character processing for DSTTYISR, DSCOMISR and DSISR402. It is entered when read interrupts are detected. This module places characters in the KSB fifo in such a way that event and Katakana characters are recognizeable to DNOS. When reads are outstanding to the port, it enters DSRTPD at the interrupt level via a BLWP for processing the Read request.

Vector Table

The vector table contains entries for discrete hardware-related functions. A subroutine call to a fixed table offset indexed by the table address is sufficient to transfer into hardware-dependent code. Vector table entries are provided for the following functions:

1. Initialize power up

2. Disconnect

3. Control half-duplex Modem

4. Select speed

5. Output 8-bit characters

6. Report carrier and Data Set Ready status

7. Read interface image

8. Write interface image

9. Start timer and schedule completion processor

10. Decode interrupt and service through state table

11. Monitor line for incoming call

12. Establish connection

13. Abort without waiting to drop RTS

10.4.5 DSRTPD Defaults.

The defaults maintained by DSRTPD correspond to the current 990 operating parameters for the terminal family and to the mode of operation used by DNOS utilities (such as SCI).

Parameters pertaining to communication line management and to modem operation default to the following set:

1. Accept incoming call.

2. Disconnect on receipt of DLE EOT.

3. Transmit even parity

4. Set receive parity bit to 0.

5. End of record character = CR

6. End of file character = EM

7. LTA character = EOT

8. Parity error substitute character = ?

10.5 ASYNCHRONOUS DSR STRUCTURE

A unique DSR structure has been designed for asynchronous device support. Information about this structure and the routines used to write a custom DSR can be found in the DNOS System Programmer's Guide. The material in this manual is implementation detail about the asynchrous DSRs. The information in the DNOS System Programmer's Guide should be read before this section.

```
OPERATING SYSTEM
      /|\
       |
       |
      \|/
 +-------------+
 | TSR         |
 | Terminal    |
 | Service     |
 | Routine     |
 |             |
 +-------------+
    |       /|\
    |        |
    |BL      |TSR
    |        |Schedule
    |        |
    |    +-------------+
    |    | ISR         |
    |    | Interrupt   |
    |    | Service     |
    |    | Routine     |
    |    |             |
    |    +-------------+
    |      |        /|\
    |    BL|         |
    |      |         |
   \|/    \|/        |
 +-------------+     |
 | HSR         |     |
 | Controller  |     | Controller
 | Service     |     | Interrupt
 | Routine     |     |
 |             |     |
 +-------------+     |
   /|\               |
    |                |
    |                |
   \|/               |
 +---------------------+
 |     CONTROLLER      |
 +---------------------+
```

Figure 10-7 DSR Structure

The asynchronus DSR design separates controller and device support into different software modules. The DSR consists of three basic modules. The controller support is provided by the HSR (Hardware controller Service Routine) module. The device support is provided by the TSR (Terminal Service Routine) module. The ISR (Interrupt Service Routine) has interrupt and high priority processing responsibility. Table 10-2 lists the basic functions of the DSR components. The functions are discussed in detail in the DNOS Systems Programmer's Guide.

Table 10-2 Asynchronous DSR Module Functions

TSR - TERMINAL SERVICE ROUTINE	* All DSR entry points (Request/Initial, Power up, Abort/Timeout, Delayed Reentry, and Interrupt Entry) * Request and completion reporting I/F to DNOS * Runs in PDT workspace * Provides software interface to device * Contains device-dependent logic
ISR - INTERRUPT SERVICE ROUTINE	* Interface to HSR for interrupt processing * High priority receive character processing * Runs in DSR interrupt workspace
HSR - CONTROLLER SERVICE ROUTINE	* Generic (subroutine) software interface to the controller hardware * Contains all controller-dependent logic * Contains all direct access to controller * Emulation of buffered controller - Hardware/Software FIFOs

There are two mechanisms for scheduling the TSR from an ISR:

* Reenter-me

* DSR priority schedule

Both of these mechanisms enter the DSR in the PDT workspace. The DSR is reentered when the next system clock interval expires if the reenter-me mechanism is invoked. Refer to the description of the reenter-me mechanism in the section on device I/O handling for further details. Another mechanism for scheduling DSR non-interrupt processing (TSR) from DSR interrupt processing (ISR) is the DSR priority schedule mechanism. This mechanism reenters the DSR after all interrupt processing for the system is complete, but before the operating system task scheduler or any task executes. This is a more direct reentry path to the DSR. It is intended only for the highest priority (non-interrupt) processing. If this mechanism is used arbitrarily, it can interfere with high priority processing of other DSRs.

Table 10-3 describes the requirements for DSR (TSR) entry points when the DSR priority schedule mechanism is used. The DSR priority schedule entry point is at relative address >14.

Table 10-3 DSR/TSR Entry Points

0000	B	@HINT	Hardware interrupt entry
0004	B	@SINT	System interrupt entry
0008	B	@PWRUP	Power-up entry
000C	B	@ABORT	I/O abort entry
0010	B	@TIMOUT	Timeout entry
0014	B	@REQUEST	Request processing
0018	B	@PRISCH	Priority scheduler entry

Refer to the description of the DSR priority schedule mechanism in the section on Device I/O Handling for further details. Other interface mechanisms between the TSR and ISR can be defined by the user within the constraints of the operating system. The interface to the HSR will be defined in a separate section. Each class contains several subroutines. These subroutines provide one or more HSR functions. The other subroutines are documented in the DNOS Systems Programmer's Guide. A list of the HSR subroutine classes are as follows:

* Power-up initialization.

* Write output signal or function.

* Read input signal or function.

* Enable/disable status change notification.

* Transmit a character.

* Write operational parameters.

* Read operational parameters.

* Request timer interval notification.

* Controller interrupt decoding.

* CI403/CI404 UART direct access.

The HSR consists of a set of subroutines. All HSR subroutines are called via branch and link (BL) instructions, and thus, 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. Status information is returned via alternate subroutine returns. The caller specifies alternate return addresses as operands of DATA assembler directives immediately following the BL subroutine call.

```
BL   @HSRSUB              Subroutine Call
DATA ALT1                 First Alternate Return
DATA ALT2                 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.

Some general register conventions are followed by HSR subroutines. In general, R0 and R10 are used as working registers by HSR subroutines. In general, these two registers are used when parameters are passed to or from the HSR. Exceptions will be noted for some HSR subroutines. R7, in most cases, is used as a pointer to the PDT.

10.5.1 Data Structures Linkage.

Figure 10-10 illustrates data structure linkages for asynchronous DSRs.

10.5.2 Data Structure Allocation.

The PDT extension is divided into two segments. One segment is physically contiguous to the PDT, and it contains data requiring the most efficient access. The other segment contains data for which the increased access time is not as great a penalty relative to 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 screen image DSRs.

All the data structures not accessed via long-distance instructions are allocated during 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 initial powerup code by the DSR. The method used for this allocation is described in the next section.

10.5.3 PDT Extension Definitions.

The section documents PDT extensions used by the asynchronous DSR to support specific devices and controllers. Software use of these data structures for a set of specific hardware is also discussed. Controller information is documented for the following set of asynchronous controllers:

* CI403/CI404 -- Four channel multiplexers
* CI401 -- Communications Interface Module
* 9902/9903 -- TMS9902 and TMS9903 based controllers:
  - S300 Base Station
  - CI421
  - CI422
  - 990/10A 9902 port
  - CI402

Extensions are also documented for the following set of peripheral devices:

* 931 and 940 VDTs
* Serial Printers

10.5.3.1 Asynchronous Local PDT Extension.

The asynchronous DSR structure requires a PDT extension as defined in Figure 10-11. Table 10-4 contains a template used by source code to reference the local PDT extension. The pathname for this template is DSALLLEX. It is available in the DNOS directory <vol>.S$OSLINK.TEMPLATE.ATABLE with the other system data structure templates. Notice that the detailed descriptions indicate a zero based index for the extension. However, in reality the extension entries will be accessed using an index relative to the beginning of the PDT as is indicated by the DORG directive of the template in Table 10-4.

This extension starts immediately after the KSB. The first two words of this extension are used to access a second DSR data structure (PDT extension) outside the local address space of the DSR. The next five words, PDXFLG through PDXCP3, are reserved for HSR use. The remaining local PDT extension words are for TSR/ISR use.

Detailed descriptions of asynchronous PDT extensions are presented in separate sections. There are descriptions for each controller type and each device type.

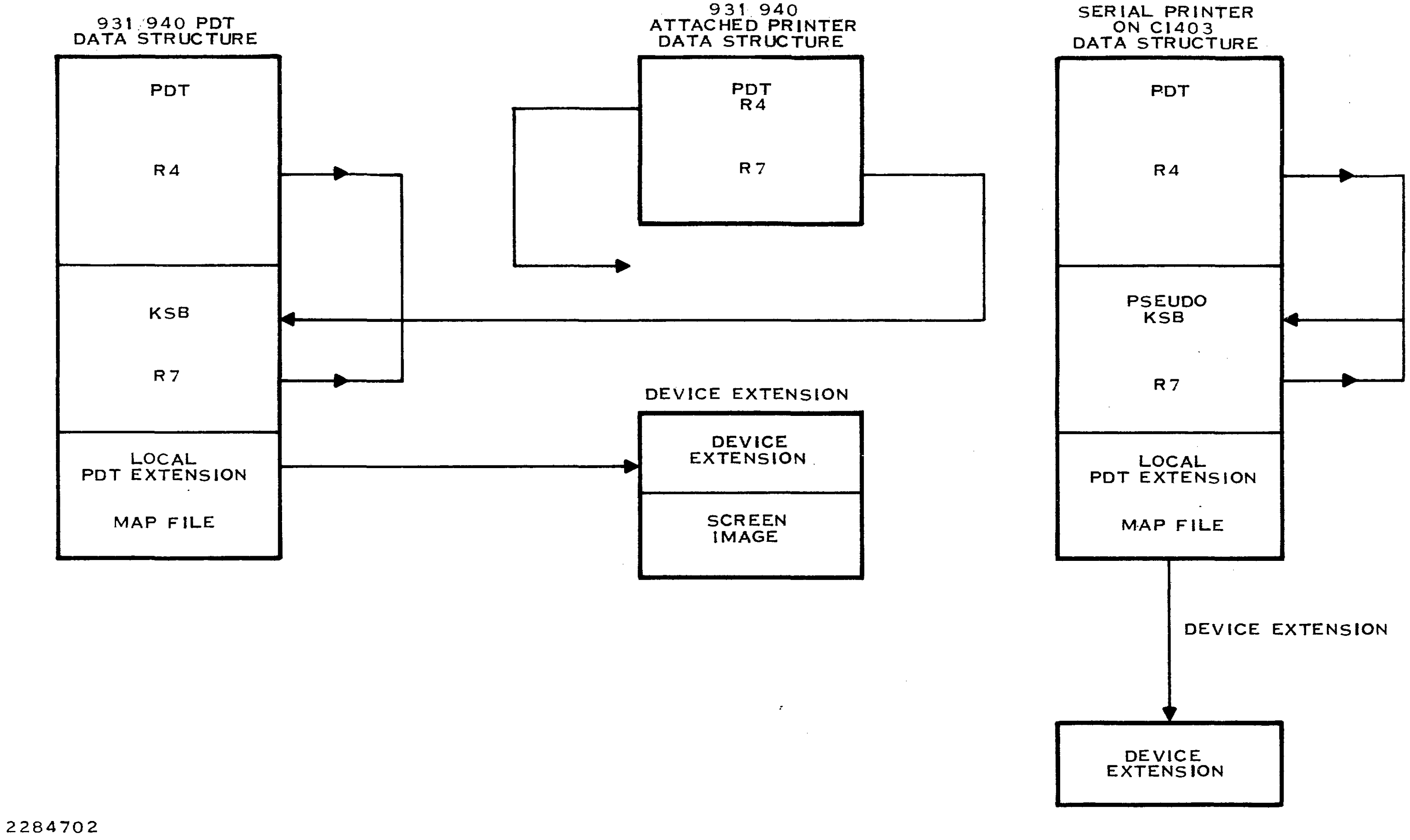


Figure 10-8 Asynchronous Data Structure Linkages

Hex. Byte	
>00	PDXSMB - LONG DISTANCE EXT. MAP BIAS
>02	PDXSMP - LONG DISTANCE EXT. MAP POINTER
>04	PDXFLG - HSR PARAMETER BYTE 0
>05	PDXCHN - HSR PARAMETER BYTE 1
>06	PDXCP1 - HSR PARAMETER BYTES 2 & 3
>08	PDXCP2 - HSR PARAMETER BYTES 4 & 5
>0A	PDXCP3 - HSR PARAMETER BYTES 6 & 7
>0C	PDXCP4 - TSR/ISR PARAMETER BYTES 0 & 1
>0E	PDXCP5 - TSR/ISR PARAMETER BYTES 2 & 3
>10	PDXCP6 - TSR/ISR PARAMETER BYTES 4 & 5
>12	PDXCP7 - TSR/ISR PARAMETER BYTES 6 & 7
>14	PDXCP8 - TSR/ISR PARAMETER BYTES 8 & 9
>16	PDXCP9 - TSR/ISR PARAMETER BYTES 10 & 11
>18	PDXCPA - TSR/ISR PARAMETER BYTES 12 & 13
>1A	PDXCPB - TSR/ISR PARAMETER BYTES 14 & 15
>1C	PDXCPC - TSR/ISR PARAMETER BYTES 16 & 17
>1E	PDXCPD - TSR/ISR PARAMETER BYTES 18 & 19
>20	PDXCPE - TSR/ISR PARAMETER BYTES 20 & 21
>22	PDXCPF - TSR/ISR PARAMETER BYTES 22 & 23

Figure 10-9 Asynchronous Local PDT Extension

Table 10-4 Asynchronous Local PDT Extension Template

```
       DORG KSBSIZ
PDXSMB BSS  2               LONG DIST EXT MAP BIAS
PDXSMP BSS  2               LONG DIST EXT MAP POINTER
PDXFLG BSS  1               HSR MEMORY AREA
PDXCHN BSS  1                     "
PDXFCT BSS  2                     "
PDXCP1 BSS  2                     "
PDXCP2 BSS  2                     "
PDXCP3 BSS  2                     "
PDXCP4 BSS  2               TSR/ISR MEMORY AREA
        "                         "
        "                         "
        "                         "
       RORG
```

10.5.3.2 Asynchronous Long-Distance Device Extension.

The asynchronous DSRs are designed to use a long-distance extension (Figure 10-12) for part of the PDT extension area. This memory must be accessed using long-distance instructions. The long-distance extension is divided into several areas as defined by the following figure (Table 10-5). The pathname for this template is DSALLREX. It is available in the DNOS directory <vol>.S$OSLINK.TEMPLATE.ATABLE with the other system data structure templates.

The first 32 bytes beginning with HSRBGN are reserved for HSR module use. The next 112 bytes provide memory for a software transmit FIFO maintained by the HSR for non-buffered controllers. (The only buffered asynchronous controllers are the CI403 and the CI404.) The remainder of the long-distance extension is for TSR/ISR use. Its size varies with the functions performed by the TSR and ISR modules. The example template defines areas for an implementation that keeps a memory copy of the screen image for VDT support. 48 bytes are for TSR/ISR use. The memory starting at SIBUFF can be used by the TSR to maintain a memory image of the CRT screen.

```
  Hex.
  Byte
        +-----------------------------------------+
  >00   |   HSRBGN - HSR PORTION OF EXTENSION     |
        |                                         |
        |              (32 BYTES)                 |
  >1F   |                                         |
        |-----------------------------------------|
  >20   |     SWFBGN - SOFTWARE FIFO BEGIN        |
        |                                         |
        |              (112 BYTES)                |
  >8F   |                                         |
        |-----------------------------------------|
  >90   | TSRBGN - TSR/ISR PORTION OF EXTENSION   |
        |                                         |
        |              (48 BYTES)                 |
  >BF   |                                         |
        |-----------------------------------------|
  >C0   |     SIBUFF - SCREEN IMAGE BUFFER        |
        |                                         |
        |             (1920 BYTES)                |
 >83F   |                                         |
        +-----------------------------------------+
```

Figure 10-10 Asynchronous Long-Distance PDT Extension

Table 10-5 Asynchronous Long-Distance PDT Extension Template

```
       DORG 0
HSRBGN EQU  $           HSR PORTION OF DEVICE EXTENSION
       BSS  >20         HSR DEPENDENT BLOCK
HSREND EQU  $
*
SWFBGN EQU  HSREND      SOFTWARE XMIT FIFO
       BSS  >70
SWFEND EQU  $
*
TSRBGN EQU  SWFEND      TSR PORTION OF DEVICE EXTENSION
       BSS  >30
TSREND EQU  $
*
SIBUFF EQU  TSREND      SCREEN IMAGE BUFFER
       BSS  >780        1920 BYTE SCREEN IMAGE BUFFER
SIEND  EQU  $
       RORG
```

10.5.3.3 CI401 HSR Local Extension.

Hex. Byte — Field Name — Description

>00 PDXSMB This word contains the inverted value of the byte count requested for the long distance buffer. The DSR initializes this word during the power-up sequence.

>02 PDXSMP This word contains the beet bias address of the long distance buffer requested by the DSR during the power-up sequence. The DSR requests the buffer by calling the system routine IOGUB.

>04 PDXFLG This byte contains bit flags for the CI401 HSR. The flags are defined as follows:

Bit 0 Controller Master Reset Failed. This flag is set to one during HSR power-up processing when a controller hardware failure is detected. This flag is also set to one if the controller is not present in the chassis. This flag is monitored by HSR interrupt processing as a means of gracefully handling controllers that are included as part of the configuration during system generation but which are not physically present in the chassis. This only becomes important when the controller in question is sharing an interrupt level with other controllers that are present in the chassis.

Bit 1 Secondary Data Carrier Detect (SDCD) State. This flag is set to one when the current state of the SDCD signal is on (logic 1) and is set to zero when the current state of the SDCD signal is off (logic 0).

Bit 2 Reserved.

Bit 3 Reserved.

Bit 4 Reserved.

Bit 5 Secondary Data Character Detect Notify Flag. This flag indicates if notification of status change has been requested. This flag is set to one when the HESSDC HSR subroutine is called to enable notification of status change. This flag is set to zero when the HDSSDC subroutine is called to disable notification of status change.

Bit 6 Reserved.

Bit 7 Reserved.

>05 PDXCHN This byte contains bit flags for the CI401 HSR. The flags are defined as follows:

Bit 0 Data Carrier Detect (DCD) State. This flag is set to one when the current state of the DCD signal is on (logic 1) and is set to zero when the current state of the DCD signal is off (logic 0).

Bit 1 Ring Indicator (RI) State. This flag is set to one when the current state of the RI signal is on (logic 1) and is set to zero when the current state of the RI signal is off (logic 0).

Bit 2 Data Set Ready (DSR) State. This flag is set to one when the current state of the DSR signal is on (logic 1) and is set to zero when the current state of the DSR signal is off (logic 0).

Bit 3 Clear To Send (CTS) State. This flag is set to one when the current state of the CTS signal is on (logic 1) and is set to zero when the current state of the CTS signal is off (logic 0).

Bit 4 Data Character Detect (DCD) Notify Flag. This flag indicates if notification of status change has been requested. This flag is set to one when the HESDCD HSR subroutine is called to enable notification of status change. This flag is set to zero when the HDSDCD subroutine is called to disable notification of status change.

Bit 5 Ring Indicator (RI) Notify Flag. This flag indicates if notification of status change has been requested. This flag is set to one when the HESRI HSR subroutine is called to enable notification of status change. This flag is set to zero when the HDSRI subroutine is called to disable notification of status change.

Bit 6 Data Set Ready (DSR) Notify Flag. This flag indicates if notification of status change

		has been requested. This flag is set to one when the HESDSR HSR subroutine is called to enable notification of status change. This flag is set to zero when the HDSDSR subroutine is called to disable notification of status change.
	Bit 7	Clear To Send (CTS) Notify Flag. This flag indicates if notification of status change has been requested. This flag is set to one when the HESCTS HSR subroutine is called to enable notification of status change. This flag is set to zero when the HDSCTS subroutine is called to disable notification of status change.
>06	PDXFCT	This word contains the entry byte count for a software transmit FIFO maintained by the HSR.
>08	PDXCP1	This word contains the insertion pointer for the software FIFO maintained by the HSR. It contains the address of the next FIFO location in which to store a transmit character.
>0A	PDXCP2	This word contains the removal pointer for the software transmit FIFO maintained by the HSR. It contains the address of the next transmit FIFO entry to be removed.
>0C	PDXCP3	This word is used by the HSR as a transmit state vector. It contains an address of an HSR transmit routine. The HSR changes this address as the HSR transmit state changes.
>0E->24	PDXCP4	Reserved for TSR/ISR usage.

10.5.3.4 CI401 HSR Long-Distance Extension.

Hex. Byte	Field Name	Description
>00	EXTTMR	HSR timer word. This word is counted down, or decremented, once every 250 milliseconds until it reaches zero. The ISR is notified of timer expiration when this value is decremented to zero. The timer count is set by calling the HSR routine HTIMER.
>02	EXTFLG	This word is used as a bit flag word by the HSR. The flag definitions are as follows:
	Bit 0	Channel transmit halt flag. When this flag

is set to one, the HSR accepts transmit data from the TSR/ISR until the software FIFO fills, but does not transmit data on the communication line. This flag is set to one by a call to the HSTCTH HSR subroutine. It is set to zero by a call to the HRTCTH HSR subroutine.

Bit 1 This flag indicates the mode of the HSR. When this flag is set to one, the channel is in a channel reset mode. A call to the HSTCR HSR subroutine sets this flag to one. Once in the channel reset mode, a call to the HRTCR (Reset channel reset mode) HSR subroutine is required to set the flag to zero and restore the HSR to normal operation. When in the channel reset mode, no CI401 interrupts are enabled.

Bit 2-F Reserved.

>04 EXTTMP This word is used as a temporary storage word by HSPPSL and HSPSPD subroutines.

>06 EXTSPD This word contains the speed selection code. It contains the value passed as a parameter to the HSPSPD subroutine. The contents of this word are returned by the HSR subroutine HRPSPD.

>08 EXTPSL This word contains the data format parameters. It contains the value of the parameters passed to the HSR subroutine HSPPSL. The contents of this word are returned to the caller of the HSR subroutine HRPPSL.

>0A EXTFL1 This word contains a receive data mask. The mask value is set by the HSR subroutine HSPPSL. The mask value is used by the HSR receive data processing routine to isolate the receive data bits.

>0C EXTFL2 This word is used as a temporary storage word by the HSR subroutine HSTCR.

>0E EXTFL3 Reserved.

>10 EXTOVR This word is used as a receive overrun error counter by the HSR.

>12 EXTFER This word is used as a receive framing error counter by the HSR.

>14 EXTPER This word is used as a receive parity error counter by the HSR.

10.5.3.5 CI403/CI404 HSR Local Extension.

Hex. Byte Field Name Description

>00 PDXSMB This word contains the inverted value of the byte count requested for the long distance buffer. The DSR initializes this word during the power-up sequence.

>02 PDXSMP This word contains the beet bias address of the long distance buffer requested by the DSR during the power-up sequence. The DSR requests the buffer by calling the system routine IOGUB.

>04 PDXFLG This byte contains bit flags for the CI403/CI404 HSR. The flags are defined as follows:

Bit 0 Transmit in hold. This flag is set to one whenever one or both of the following conditions exists: the common transmit FIFO is near full, or the channel-specific transmit FIFO is full. This flag bit 0 is reset whenever all holding conditions have been satisfied. See the explanation below for bits 2 and 3.

Bit 1 Reserved.

Bit 2 Controller transmit hold. Whenever the common transmit FIFO is near full, the controller issues the halt transfer status (status 4, substatus 1) to the HSR; this bit is set to one, and bit 0 is set to one. Bit 2 is reset when the common transmit FIFO is empty and the controller issues the resume transfer status (status 4, substatus 2) to the HSR.

Bit 3 FIFO exhausted transmit hold. Whenever the HSR FIFO count goes to zero, the HSR sets this bit to one, sets bit 0 to one, and enables transmit FIFO empty interrupt. The HSR resets bit 3 whenever the controller notifies the HSR that the channel specific FIFO is empty.

Bit 4 Data Character Detect (DCD) notification flag. This bit is set to one when the TSR/ISR calls the HSR subroutine HESDCD to enable notification of DCD status changes. This bit is set to zero when the HDSDCD subroutine is called to disable notification of DCD status changes.

Bit 5 Ring Indicator (RI) change notification flag. This bit is set to one when the TSR/ISR calls the HSR subroutine HESRI to enable notification of RI status changes. This bit is set to zero when the HDSRI subroutine is called to disable notification of RI status changes.

Bit 6 Data Set Ready (DSR) change notification flag. This bit is set to one when the TSR/ISR calls the HSR subroutine HESDSR to enable notification of DSR status changes. This bit is set to zero when the HDSDSR subroutine is called to disable notification of DSR status changes.

Bit 7 Clear To Send (CTS) change notification flag. This bit is set to one when the TSR/ISR calls the HSR subroutine HESCTS to enable notification of CTS status changes. This bit is set to zero when the HDSCTS subroutine is called to disable notification of CTS status changes.

>05 PDXCHN This byte contains the channel number of the CI403/CI404. It is specified during system generation and initialized by the system generation program.

>06 PDXFCT This word contains the byte count of the available CI403/CI404 channel-specific transmit FIFO as maintained by the HSR. This word is not necessarily an accurate representation of the actual state of the controller FIFOs.

>08 PDXCP1 This word contains bit flags for the CI403/CI404 HSR. The flags are defined as follows:

Bit 0 Reset mode. This flag is set to one when the TSR/ISR calls the HSR subroutine HSTCR to reset a particular channel. While this bit is set to one, all controller interrupts are ignored. This bit is reset to zero when the

		TSR/ISR calls the HSR subroutine HRTCR to allow controller interrupts.
	Bit 1	Secondary Data Character Detect (SDCD) change notification flag. This bit is set to one when the TSR/ISR calls the HSR subroutine HESSDC to enable notification of SDCD status changes. This bit is set to zero when the HDSSDC subroutine is called to disable notification of SDCD status changes.
	Bit 2-r	Reserved.
>0A	PDXCP2	Reserved.
>0C	PDXCP3	Reserved.
>0E->24	PDXCP4	Reserved for TSR/ISR usage.

10.5.3.6 CI403/CI404 HSR Long-Distance Extension.

Hex. Byte	Field Name	Description
>00	EXTTMR	HSR timer duration value. This word is decremented once every 250 milliseconds until it reaches zero. The TSR/ISR is notified of timer expiration when this value is decremented to zero. The timer count is set by calling the HSR subroutine HTIMER.
>02	EXTRG3	This word contains a copy of the ACE register 3 contents for the specified channel.
>04	EXTRG4	This word contains a copy of the ACE register 4 contents for the specified channel.
>06	EXTRG7	This word contains a copy of the ACE register 7 contents for the specified channel.
>08	EXTSPD	This byte contains a copy of the speed code that is currently programmed in the ACE. If the TSR/ISR specified an illegal speed then this byte contains an >FF. The second byte of the word is reserved.
>0A	EXTR0	This word contains a copy of the TSR's R0 whenever the HSR is delaying after a write to an ACE register.
>0C	EXTR7	This word contains a copy of the TSR's R7 whenever the HSR is delaying after a write to an ACE register.

>0E	EXTR11	This word contains a copy of the TSR's R11 whenever the HSR is delaying after a write to an ACE register.

10.5.3.7 9902/9903 HSR Local Extension.

Hex. Byte	Field Name	Description
>00	PDXSMB	This word contains the inverted value of the byte count requested for the long distance buffer. The DSR initializes this word during the power-up sequence.
>02	PDXSMP	This word contains the beet bias address of the long distance buffer requested by the DSR during the power-up sequence. The DSR requests the buffer by calling the system routine IOGUB.
>04	PDXFLG	This byte contains bit flags for the 9902/9903 HSR. The flags are defined as follows:
	Bit 0	Data Carrier Detect (DCD) State. This flag is set to one when the current state of the DCD signal is on (logic 1), and is set to zero when the current state of the DCD signal is off (logic 0).
	Bit 1	Ring Indicator (RI) State. This flag is set to one when the current state of the RI signal is on (logic 1), and is set to zero when the cur- rent state of the RI signal is off (logic 0).
	Bit 2	Data Set Ready (DSR) State. This flag is set to one when the current state of the DSR signal is on (logic 1), and is set to zero when the current state of the DSR signal is off (logic 0).
	Bit 3	Clear To Send (CTS) State. This flag is set to one when the current state of the CTS signal is on (logic 1), and is set to zero when the current state of the CTS signal is off (logic 0).
	Bit 4	Data Character Detect (DCD) Notify Flag. This flag indicates if notification of status change has been requested. This flag is set to one when the HESDCD HSR subroutine is called to enable notification of status

change. This flag is set to zero when the HDSDCD subroutine is called to disable notification of status change.

Bit 5 Ring Indicator(RI) Notify Flag. This flag indicates if notification of status change has been requested. This flag is set to one when the HESRI HSR subroutine is called to enable notification of status change. This flag is set to zero when the HDSRI subroutine is called to disable notification of status change.

Bit 6 Data Set Ready (DSR) Notify Flag. This flag indicates if notification of status change has been requested. This flag is set to one when the HESDSR HSR subroutine is called to enable notification of status change. This flag is set to zero when the HDSDSR subroutine is called to disable notification of status change.

Bit 7 Clear To Send (CTS) Notify Flag. This flag indicates if notification of status change has been requested. This flag is set to one when the HESCTS HSR subroutine is called to enable notification of status change. This flag is set to zero when the HDSCTS subroutine is called to disable notification of status change.

>05 PDXCHN This byte contains bit flags for the 9902/9903 HSR. The flags are defined as follows:

Bit 0 Reserved.

Bit 1 Secondary Data Carrier Detect (SDCD) State. This flag is set to one when the current state of the SDCD signal is on (logic 1), and is set to zero when the current state of the SDCD signal is off (logic 0).

Bit 2 Transmit Shift Register Empty (TSRE) State. This flag is set to one when the current state of the TSRE signal is on (logic 1), and is set to zero when the current state of the TSRE signal is off (logic 0).

Bit 3 Reserved.

Bit 4 Reserved.

	Bit 5	Secondary Data Character Detect Notify Flag. This flag indicates if notification of status change has been requested. This flag is set to one when the HESSDC HSR subroutine is called to enable notification of status change. This flag is set to zero when the HDSSDC subroutine is called to disable notification of status change.
	Bit 6	Transmit Shift Register Empty (TSRE) Notify Flag. This flag indicates if notification of status change has been requested. This flag is set to one when the HESTSR HSR subroutine is called. This flag is set to zero when the HDSTSR subroutine is called.
	Bit 7	Reserved.
>06	PDXFCT	This word contains the entry byte count for a software transmit FIFO maintained by the HSR.
>08	PDXCP1	This word contains the insertion pointer for the software FIFO maintained by the HSR. It contains the address of the next FIFO location in which to store a transmit character.
>0A	PDXCP2	This word contains the removal pointer for the software transmit FIFO maintained by the HSR. It contains the address of the next transmit FIFO entry to be removed.
>0C	PDXCP3	This word is used by the HSR as a transmit state vector. It contains an address of an HSR transmit routine. The HSR changes this address as the HSR transmit state changes.
>0E->27	PDXCP4	Reserved for TSR/ISR usage.

10.5.3.8 9902/9903 HSR Long-Distance Extension.

Hex. Byte	Field Name	Description
>00	EXTTMR	HSR timer word. This word is decremented once every 250 milliseconds until it reaches zero. The ISR is notified of timer expiration when this value is decremented to zero. The timer count is set by calling the HSR routine HTIMER.
>02	EXTCDY	HSR timer word. This word is decremented approximately once every 16 milliseconds

until it reaches zero. The EXTTMR word is then decremented, and this word is restored from EXTCST.

>04 EXTCST HSR timer word. This word is loaded at initial DSR power-up entry to be used to determine how many 9902/9903 timer interrupts are required to time 250 milliseconds.

>06 EXTFLG This word is used as a bit flag word by the HSR. The flag definitions are as follows:

Bit 0 Channel transmit halt flag. When this flag is set to one the HSR accepts transmit data from the TSR/ISR until the software FIFO fills, but does not transmit data on the communication line. This flag is set to one by a call to the HSTCTH HSR subroutine. It is set to zero by a call to the HRTCTH HSR subroutine.

Bit 1 This flag indicates the mode of the HSR. When this flag is set to one the channel is in a channel reset mode. A call to the HSTCR HSR subroutine sets this flag to one. Once in the channel reset mode, a call to the HRTCR (Reset channel reset mode) HSR subroutine is required to set the flag to zero and restore the HSR to normal operation. When in the channel reset mode, no 9902/9903 interrupts are enabled.

Bit 2 UART Internal Loopback Enabled Flag. This flag is set to one when the TSR requests that the 9902/9903 chip be placed in UART loopback mode. This bit is set to one by a call to the HSTUIL HSR subroutine. It is set to zero by a call to the HRTUIL HSR subroutine.

Bit 3 Transmit Break Enabled Flag. This flag is set to one when the TSR requests that the 9902/9903 chip transmit a break condition. This bit is set to one by a call to the HSTTB HSR subroutine. It is set to zero by a call to the HRTTB HSR subroutine.

Bit 4-F Reserved.

>08 EXTTMP This word is used as a temporary storage word by the HSR.

>0A EXTTM1 This word is used as a temporary storage word by the HSR.

>0C EXTSPD — This word contains the speed selection code. It contains the value passed as a parameter to the HSPSPD subroutine. The contents of this word are returned by the HSR subroutine HRPSPD.

>0E EXTPSL — This word contains the data format parameters. It contains the value of the parameters passed to the HSR subroutine HSPPSL. The contents of this word are returned to the caller of the HSR subroutine HRPPSL.

>10 EXTLDC — This word contains the CRU instruction required to load the 9902/9903 UART with transmit data for the communications line.

>12 EXTTYP — This word is set up at initial DSR entry on power-up, to define the hardware interface type as follows:

Value	Port	Device Type
0	990/10A 9902	>0007
2	CI402 9902	>0008
4	CI421 9902	>0009
6	CI422 9902	>000A
8	S300 Base Station 9902	>0006
>A	CI421 9903	>0030

>14 to >BF — These words are reserved for future HSR development.

10.5.3.9 931/940 TSR/ISR Local Extension.

Hex. Byte — Field Name — Description

>00 PDXSMB — This word contains the inverted value of the byte count requested for the long distance buffer. The DSR initializes this word during the power-up sequence.

>02 PDXSMP — This word contains the beet bias address of the long distance buffer requested by the DSR during the power-up sequence. The DSR requests the buffer by calling the system routine IOGUB.

>04->0D — HSR Information.

>0E PDXCP4 — System generation puts in this word the location of R0 of the attached printer if one is present. At power-up time, this is moved

to the remote extension, and the word is used as the TSR's copy of the IRB extended user flags.

>10 PDXCP5 System generation puts in this word the speed of the communication line and a flag indicating if the line is a dial-in line. At power-up time, this is moved to the remote extension, and the word is used as the current cursor position by the TSR.

>12 PDXCP6 This word is a TSR flag word.

Bit 0 Terminal Type. This flag is set to one if the terminal is a 931 and set to zero if it is a 940.

Bit 1 Read Schedule. This flag is set to one when the TSR requests to be resecheduled upon receipt of a read character.

Bit 2 Printer Has Channel. This flag is set to one when the printer has control of the channel.

Bit 3 Printer Wants Channel. This flag is set to one when the printer requests control of the channel, and the CRT currently has control of the channel.

Bit 4 CRT Has Channel. This flag is set to one when the CRT has control of the channel.

Bit 5 CRT Wants Channel. This flag is set to one when the CRT requests control of the channel, and a printer currently has control of the channel.

Bit 6 Cursor Is On. This flag is set to one when the cursor is on.

Bit 7 Cursor Not Been Moved. This flag indicates that the cursor is not physically at the location that the internal pointer says it is located. This is to avoid sending unnecessary commands.

Bit 8 Graphics Flag. This flag indicates that the command has been made to the terminal to be in graphics mode.

Bit 9 Graphics Input Flag. This flag indicates that the terminal has notified the host that input from the terminal will be in graphics mode.

Bit 10 Reserved.

Bit 11 Clear Flag. This flag indicates that the screen has been cleared and nothing has yet been sent to it. This is to avoid sending unnecessary commands.

Bit 12 Cursor is Blinking. This flag is set to one when the cursor is blinking.

Bit 13 Initialized Flag. This flag indicates that the terminal has been initialized.

Bit 14 Extent Flag. This flag indicates that the command has been made to the terminal to define the end of the current field for insert and delete.

Bit 15 Insert Flag. This flag indicates that the TSR is in insert mode.

>14 PDXCP7 This word is used as an internal buffer address by the TSR.

>16->1A PDXCP7-A These words are temporary save locations for the TSR.

>1C PDXCPB This word contains the current attribute sent to the terminal.

>1E PDXCPC This word contains the counter for the optimization routine.

>20 PDXCPD This word contains an internal TSR counter.

>22 PDXCPE This word is an opcode 15 flag.

Bit 0 Pass Through. This flag is set to one when the data is to be sent and received with no conversion. The only characters that are acted upon are DC1 and DC3 (ready/busy).

Bit 1 ETX Flag. Terminate a Pass Through Read on receipt of an "ETX" character.

Bit 2 ESC Flag. Terminate a Pass Through Read on receipt of an "ESC)" character string.

Bit 3 Extended Event Flag. This flag allows the extra 940 characters to be entered.

Bit 4 Reserved.

Bit 5 Special Attribute Flag. This flag is set to one when the the TSR will allow "SI", "SO", or "ESC 4" to be sent to the terminal from a user buffer.

Bit 6 Disable Attributes. This flag means that no attributes are to be sent to the terminal by the TSR.

Bit 7 Reserved.

Bit 8 Modified Flag. This flag indicates that if any data has been modified in a field, the task should be notified.

Bit 9 Extended Character Validation. This flag indicates that the character validation is to be handled before the character is echoed.

Bit 10 Null Truncation flag.

Bits 11-15 Reserved.

>24 PDXCPF This word contains the current state of input as follows:

Value	Meaning
0	Ignore input and do not produce output.
4	Accept input and produce output.

10.5.3.10 931/940 TSR/ISR Long-Distance Extension.

Hex. Byte	Field Name	Description
>00->8F		HSR Words.
>90	VDTFIL	This byte contains the current fill character.
>91	VDTEVT	This byte contains the event character that terminated the read.
>92	VDTDEF	This word contains the sequential cursor position of the beginning of the current field.
>94	VDTPSR	This word is a temporary location for the printer portion of the TSR.

>96 VDTPTP This byte contains the current printer type as follows:

Value	Meaning
0	150 cps
1	75 cps
2	40 cps
3	20 cps
4	300 cps

>97 Reserved.

>98 Run-time location for speed in PDXCP5.

>9A Mask to determine if the terminal is connected.

Value	Meaning
>A000	DCD and DSR
>2000	DSR only

>9C Remote Flags:

Bit 0 Reserved.

Bit 1 Blinking. This flag is set if blinking is allowed for the terminal (940 only).

Bit 2 Wait for Positive Feedback. This flag is set when a printer buffer has been sent and the TSR is waiting for terminal acknowledgement (931 only).

Bit 3 Schedule on Positive Feedback. This flag is set when a printer buffer has been sent and the TSR wants to be scheduled when the terminal acknowledges the buffer (931 only).

Bit 4 Terminal has started power-up, but has not completed yet.

Bit 5 An immediate open has occurred on the CRT, the next close will be an immediate close.

Bit 6 An immediate open has occurred on the printer, the next close will be an immediate close.

	Bit 7-D	Reserved.
	Bit E	ESC Found. This flag indicates that an ESC was just found in a string in the TSR.
	Bit F	Reserved.
>9E	VDTERR	This word is used to pass an error code from the ISR to the TSR.
>A0	VDTOVR	This word is a counter of the number of overrun errors detected.
>A2	VDTPAR	This word is a counter of the number of parity errors detected.
>A4	VDTFRM	This word is a counter of the number of framing errors detected.
>A6	VDTPTR	Run-time location for printer address in PDXCP4.
>A8	PRTTIM	Timer for printer delay (940 only).
>AA	VDTEDL	This word is an opcode 15 Edit Flag.
	Bit 0	Erase Field is an Event.
	Bit 1	Right Field is an Event.
	Bit 2	Cursor Left out of a Field is an Event.
	Bit 3	Tab is an Event.
	Bit 4	Reserved.
	Bit 5	Skip is an Event.
	Bit 6	Home is an Event.
	Bit 7	Return is an Event.
	Bit 8	Erase Input is an Event.
	Bit 9	Reserved.
	Bit A	Delete Character is an Event.
	Bit B	Insert Character is an Event.
	Bit C	Cursor Right out of a Field is an Event.
	Bit D	Enter is an Event.

Bit E Left Field is an Event.

Bit F Reserved.

>AE STATBD This word contains the state of the board as follows:

Value	Meaning
0	Board disconnected.
4	Board connected.
8	Board waiting on timeout.
12	Board waiting on ring.
16	Board is in DSR diagnostic mode.

>B0 STATGR This word contains the state of the input graphics.

Value	Meaning
0	Input is not graphics.
4	Input is graphics.

>B2 STATFN This word contains the state of the key mapping.

Value	Meaning
0	Regular keys.
4	ESC was received.
8	Aid was received.
12	Pass Through Mode.
16	DSR Diagnostic Mode.
20	Read Status Mode.

10.5.3.11 Serial Printer HSR Local Extension.

Hex. Byte	Field Name	Description
>00	PDXSMB	This word contains the inverted value of the byte count requested for the long distance buffer. The DSR initializes this word during the power-up sequence.
>02	PDXSMP	This word contains the beet bias address of

the long distance buffer requested by the DSR during the power-up sequence. The DSR requests the buffer by calling the system routine IOGUB.

>04 - >0F Reserved for HSR use.

>10 PDXCP4 This byte contains bit flags for the serial printer HSR. The flags are defined as follows:

Bit 0 Reserved.

Bit 1 This bit, when set to one, indicates that a write SCB is being processed by the TSR.

Bit 2 Reserved.

Bit 3 This bit, when set to one, indicates that a non-write command is being processed by the TSR.

Bit 4 This bit, when set to one, indicates that data transmission to the device has been stopped because the data set ready (DSR) signal is not present.

Bit 5 This bit, when set to one, indicates that data transmission to the device has been stopped due to reception of a DC3 character from the device.

Bit 6 This bit, when set to one, indicates that a KATAKANA mode select character is being transmitted.

Bit 7 This bit, when set to one, indicates that the device supports extended print (lower case letters).

Bit 8 Reserved.

Bit 9 This bit, when set to one, indicates that the ISR section has detected a condition that requires scheduling of the TSR section.

Bit 10 Reserved.

Bit 11 Reserved.

Bit 12 This bit, when set to one, indicates that the TSR is not able to operate due to some condition (hardware interface not present,

		requested baud rate not supported by the hardware interface, power up failure, and so on).
	Bit 13	This bit, when set to one, indicates that initial power-up has occurred.
	Bit 14	Reserved.
	Bit 15	Reserved.
>12	PDXCP5	This word contains the transmit and receive baud rate word. This word is initialized by sysgen or the MVPC command.
>14	PDXCP6	This word contains the operation parameter word for the hardware interface.
>16	PDXCP7	This word contains the total error count seen by the TSR (the count of all parity errors, overrun errors, framing errors, and so on).

10.5.3.12 Serial Printer HSR Long-Distance Extension.

Hex. Byte	Field Name	Description
>00 - >8F		These words are reserved for use by the HSRs.
>90	RTEXT0	This word is a count of the number of receive break conditions sensed.
>92	RTEXT1	This word is a count of the number of framing errors received.
>94	RTEXT2	This word is a count of the number of parity errors received.
>96	RTEXT3	This word is a count of the number of receiver overrun errors received.
>98	RTEXT4	This word is a count of the number of other errors received.
>9A	RTEXT5	This word is a count of the number of illegal UART interrupts received.
>9C - >BF		These words are reserved for future ISR/TSR use.

10.6 I/O UTILITY (IOU)

The initial processing of a utility request is similar to that for device I/O. The SVC is decoded by RPROOT, which passes control to IOPREP, the I/O preprocessor. IOPREP passes control to the I/O Utility preprocessor, IUPREP, when it recognizes an IOU SVC.

The IOU preprocessor buffers the call block and any extensions to the call block as required by the individual subopcodes. Call block extensions include pathname, key definitions, and parameters. The preprocessor checks for illegal subopcodes and mapping errors during the buffering process. BRBs for SVCs that do not require an access name (or whose access name begins with a period) are placed directly on the input queue for the IOU task. Other BRBs are first queued for the Name Manager task for resolution of any logical names. After logical name resolution, Name Manager places the BRB on the input queue for the IOU task.

The IOU task has a main driver that calls other routines to process the individual IOU subopcodes. The IOU task consists of a root segment and three or four system overlays depending upon the sysgen options chosen. Assign LUNO, Release LUNO, and Delete File SVCs are handled by code in the IOU root segment. Create File SVCs are handled by code residing in one overlay; Add Alias, Delete Alias, and Rename <file in a second overlay; and the remaining IOU SVCs are handled by code in the third overlay. A fourth overlay to handle security is included if the security option was chosen during sysgen. File security will be discussed in a separate section. The main driver, a stack, and subroutines required by all overlays are included in the IOU root segment.

When IOU is activated, it dequeues an entry from its queue and processes the entry. If the IOU SVC processor that handles the SVC resides in an overlay, the overlay is loaded into memory (unless it already resides in memory). The SVC processor is called to perform the required operations; when it completes, control is returned to the IOU main driver. The processing of the SVC may require that the BRB be queued for processing by a channel owner task. If so, the request is passed to the IPÇ preprocessor for routing to the channel owner.

The IOU main driver takes the appropriate exit path after the SVC processor has handled the SVC. If the request was queued to a channel owner, the BRB will be unbuffered later, after it has been processed by the channel owner and placed back on the IOU input queue by IPC. If the SVC required the creation of a job temporary file, the BRB is placed on the input queue for Name Manager for final processing and unbuffering. If neither of these special conditions exists, the BRB is queued for unbuffering to the calling task. IOU then requests the next entry from its queue. If the input queue is empty, IOU suspends, awaiting queue input.

The range of IOU operations includes those that process SVC requests to create and delete files, assign and release LUNOs, rename files, protect and unprotect files, add and delete aliases for filenames, and create and delete channels. IOU also processes requests for special operating system services such as releasing LUNOs in another job.

10.6.1 Configurability.

IOU is configurable during sysgen when the user specifies the features desired. The modules to process these features are then selected for inclusion in the IOU task. The following feature levels can be selected:

* Dynamic KIF creation and deletion

* File access security

10.6.2 Memory Layout.

IOU runs as a system task in map file 1. Its three map segments are used as follows:

* The first segment contains the system root.

* The second segment changes during processing and contains either the user JCA, the system job JCA, or a file management table area (FMT).

* The third segment contains the IOU code.

10.6.3 Structures Maintained by IOU.

The file management table areas (FMTs) are memory resident segments, the number and size of which are defined during sysgen. The FMT is used primarily for in memory representation of files currently in use. One may monitor the usage of the FMT by using the Execute Performance Display (XPD) command. If the system is approaching maximum utilization, the error message INSUFFICIENT FILE MANAGEMENT TABLE AREA AVAILABLE will be issued. In any case, if it is determined that more table area is needed, the Modify System Table (MST) command may be issued to increase the FMT size up to 256K bytes.

The following structures are built by or used by IOU, and they are found in several different areas, as noted in the descriptions. Detailed descriptions of the structures are shown in the section on data structure pictures.

* Directory overhead record (DOR) - Disk structure that shows the number of files in a directory, the number of available entries, the number of temporary files, and several other pieces of miscellaneous directory data.

* File descriptor record (FDR) - Disk directory record that describes the name, location, and characteristics of a disk file and the ID of the user who created the file.

* Alias descriptor record (ADR) - Disk structure that contains an alternate pathname component for some file pathname and points to the FDR for which this name is an alias.

* Key descriptor record (KDR) - Disk structure that describes the keys of a key indexed file; a KDR for a multifile KIF set contains additional fields to detect illegal attempts to combine files.

* Channel descriptor record (CDR) - Disk directory record that describes the name and characteristics of an IPC channel. The CDR is located in the directory containing the FDR of the program file that contains the channel owner task, and it contains the record number of that FDR.

* File structure common (FSC) - Template to define the FDB and FCB variants.

* File directory block (FDB) - A single node of the in-memory directory tree structure located in the file management table areas. Provides tree linkage and the information required to perform direct disk I/O to a directory. An FDB is created for each pathname component on an assign or attach operation. The FDB is a variant of the FSC structure.

* File control block (FCB) - In-memory equivalent of the FDR, located in the file management table area. The FCB is built for only the last component of a file pathname. The FCB points to its corresponding FDB. The FCB is a variant of the FSC structure.

* Resource ownership block (ROB) - The ROB represents a job's ownership of a file resulting from an Attach Resource operation. The ROB points to an FCB. ROBs are always built in the JCA and are chained in a list

anchored by JITROB.

* Channel control block (CCB) - In-memory equivalent of a CDR, built on an Assign LUNO to an IPC channel. CCBs contain a pointer to the File Descriptor Packet (FDP) of the program file containing the channel owner task and the eight-character name of the last component of the channel pathname. CCBs for global channels are built in the STA, and are anchored from CCBSTR in NFPTR. CCBs for job-local and task-local channels are chained in a single list anchored from JITCCB in the JCA. The CCB points to the JSB and TSB of the owner task.

* Logical device table (LDT) - The LDT includes a description of an I/O resource, usage flags, ownership information, and file rights. Job-local and task-local LDTs are built in the JCA and are anchored from JITLDT and TSBLDT, respectively. Global LDTs are built in the STA and are anchored from LDTLST in NFPTR. LDTRLK points to a CCB, FCB, or PDT, depending on the LUNO assignment.

* Resource privilege block (RPB) - The RPB is a structure used to control access privileges for resources. The RPB is built on each Assign LUNO to a device, file, or IPC channel. The RPB contains the open access privilege flags (2 bits), the address of its associated LDT, and the JSB address of the job that assigned the LUNO (the JSB field is zero for global LUNOS). An RPB for a device is chained to the PDT, an RPB for a channel is linked to the CCB, and an RPB for file is chained to the FCB. The RPB contains currency information for LUNOs assigned to files.

10.6.3.1 Directory Tree Construction.

In DNOS, an FDB is built in the file management table area for each component of a file pathname, and an FCB is built there for the last component of the pathname.

Each disk PDT extension for a file contains the address of the VCATALOG FDB and the SSB address of the file management table area in which it resides. The VCATALOG entry is placed there by IPL for all devices which are on line during IPL. They are placed there by Install Volume (IV) otherwise. IOU maps this area into its second segment to begin the tree search. Each pointer to a node of the tree contains the SSB address of the file management table area where that node resides. Since each FDB could potentially reside in a different table area, IOU must be sure that it has the correct segment mapped in before following an FDB pointer.

When a new node is to be added to the tree, an attempt is made to allocate it in the same table area as its parent. If no space remains, other file management table areas are checked, if any exist. FMSTR in NFPTR contains the SSB address of the first file management table area, and FMEND contains the SSB address of the last. The SSBs for the table areas are chained together. IOU maps in each successive table area and attempts to obtain space for the FDB. The requester is given an error if no table space is available. When linking a new FDB into the tree, IOU must be sure it has the correct table mapped in when changing parent and sibling FDB linkages.

For files to which global LUNOs are assigned, the LDT is built in the STA.

10.6.3.2 LDT Structure.

The LDT is composed of two parts built by IOU when a LUNO is assigned to an I/O resource. These parts are the LDT and an RPB. The LDT is linked into the LDT chain. The RPB for a file is allocated in the same segment as the FDB of the corresponding node. It is impractical to chain together all LDTs assigned to a file because the LDTs may be distributed to various JCAs. The RPB chain is the chain that can be traversed to find all users of a given resource. Figure 10-13 shows typical LDT chains for a task that is associated with a station and a task that is not associated with a station.

When a LUNO is opened, IOPREP searches the RPB chain to check for access privilege conflicts. Each RPB contains a two-bit field representing its access privileges and a flag indicating an open LUNO. A privilege conflict could occur with other open LUNOs. If no conflicts arise, the access privileges are recorded in the RPB and it is marked open.

The RPB contains currency information (including the current file index set up by file management for concatenated files). Currency for any LUNOs assigned to a file may be updated by File Management as necessary by updating the currency in the RPB. (Note that this is not the same as the KIF currency information.)

Parameters may be included in the parameter field of an IOU operation. The following parameters may be included, organized in the sublists shown. See the format of parameters described in the Name Management paragraph in the section on Special SVCs.

Sublist Type	Parameter Number	Parameter Description
00	03	Job access level
	04	File type
	05	Job local temporary file
	06	Initial file allocation
	07	Secondary allocation
	08	Logical record length
	09	Physical record length
	0D	Expandable
	0E	Forced write
	0F	Blank suppressed
	10	Max # of tasks
	11	Max # of procedures
	12	Max # of overlays
	14	Max # of directory entries
	15	Default physical record size
	16	KIF definition block
02	--	User ID parameter (see UIP template)
04	--	Modify file name security option
05	--	Continue LAN session on release Luno

Each of the parameters is optional and may or may not have been specified by the user. If a system parameter is chosen (those with a sublist # of 0), the parameter overrides what is given in the call block. System parameters are used to communicate information between NAMMGR and IOU. They are not intended for general use. Parameters 2, 4 and 5 are described in more detail in the SVC manual.

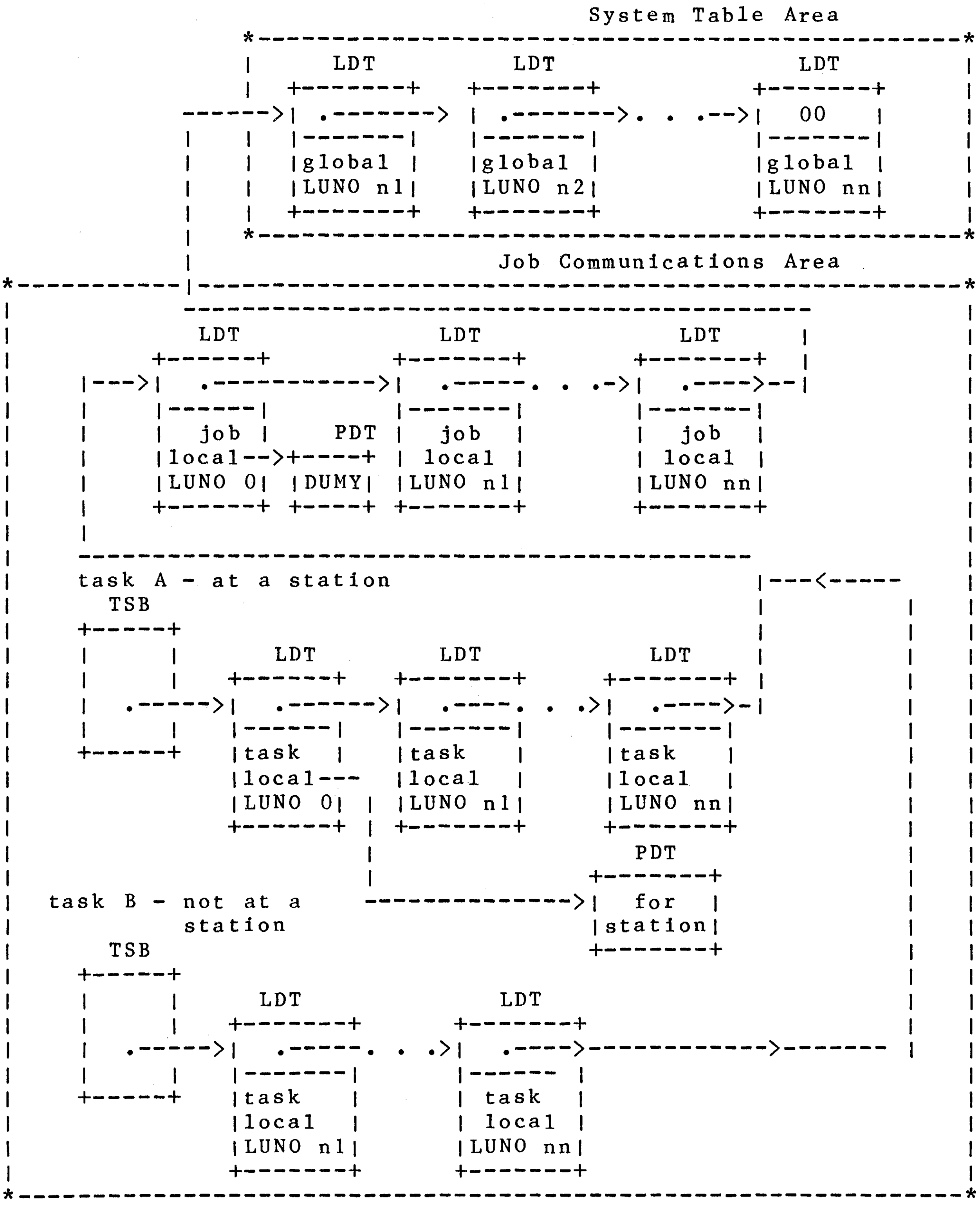


Figure 10-11 LDT Chains

10.6.4 Details of IOU Processing.

IOU processing occurs in a number of modules. Those described here include the preprocessor, IUPREP, the IOU task, and a number of modules that support special functions.

10.6.4.1 IOU Preprocessor (IUPREP).

IUPREP runs in map file 0 as XOP-level code. The call block is buffered according to the format required by each subopcode. IOU processes the following subopcodes. (Starred codes are NOT documented in user manuals; they are to be used only by operating system tasks.)

	90	Create File
	91	Assign LUNO to Pathname
	92	Delete File
	93	Release LUNO from Pathname
*	94	Assign Diagnostic Device
	95	Rename File (Assign New Filename)
	96	Unprotect File
	97	Write Protect File
	98	Delete Protect File
	99	Verify Pathname
	9A	Add Alias
	9B	Delete Alias
	9C	Define Forced Write Mode
	9D	Create IPC Channel
	9E	Delete IPC Channel
*	A0	Attach Resource
*	A1	Detach Resource
*	A3	Detach Resource by Number
*	A4	Modify FDR Bit
*	A5	Release LUNO in Another Job
*	A6	Assign System LUNO FF
*	A7	Release File Structures

* Not documented to users.

Pathname characters must be in the following ranges to allow for international character support by DNOS:

>24, >28, >29, >2E, >30 through >39, >41 through >5A,
>61 through >7A (standard English pathnames)

>5B through >5D (European characters)

>A6 through >DF (Katakana characters)

Pathname length is checked for all subopcodes that have a pathname. If the length is zero or is greater than 48, error code >92 (Bad Pathname Syntax) is set in the user call block, and an exit is made to RPRTNE in RPROOT.

The module IUVRFY is used to verify pathname syntax. It can be linked with any code that performs pathname verification. When the routine is called, register 1 must have the address of the buffer containing the pathname. The buffer has the length of the pathname in its first byte. Register 10 must be a seven word buffer to be used as a stack by IUVRFY. Register 0 will be modified by IUVRFY, but no other registers of the calling code will be modified. If the pathname is correct in syntax, register 0 receives a value >0000. If the pathname is incorrect, register 0 receives >9200.

There are two entry points in IUVRFY. IUVPND is used if the pathname can contain both upper and lower case letters, numerals, the dollar sign, periods, left and right parentheses around the last pathname element, and the pound sign(#). The katakana character set and special European characters are also permitted. The entry point IUVLET is used if the pound sign is not permitted in the pathname. Entry to the routine is via a branch and link (BL) to IUVPND or IUVLET, with a data word following the BL instruction containing the return address for error conditions.

The IOU preprocessor defines an RDB that specifies the buffering of the standard IOU call block. IUPREP builds RDB expansions dynamically to specify additional buffering as follows:

- If the subopcode requires a pathname, it is set up to be buffered in the STA along with the call block, with the pathname pointer placed in the BRB.
- A flag in IRBFLG indicates whether the IRB parameter pointer is valid on Assign LUNO operations. If it is valid, the parameter list is buffered into the caller's JCA and the parameter pointer is set in the BRB. The use of parameters on Assign LUNO is for the operating system only and is not documented in user manuals.
- If the utility flags specify KIF, either of the following may apply:
  - If the subopcode is >91 (Assign Luno) and either the temporary file bit or the autocreate bit is set, buffer the keys to the STA and set a pointer in the BRB.
  - If the subopcode is >90 (Create File), buffer the keys to the STA and set a pointer in the BRB.

A call is issued to the request processor buffering routine, RPBUF, to perform the buffering as defined by the RDB and RDB expansion definitions.

BRBs for all subopcodes that have pathnames are placed on the Name Manager queue, unless the first character of the pathname is a period. In the last case, the BRB is placed directly on the queue for the IOU task.

Name Manager checks to see if the access name is a logical name. If so, the true pathname(s) are buffered into the STA, and the buffered pathname pointer is reset. Name Manager may add parameters to the IRB.

If parameters exist on an Assign LUNO, they are buffered into the STA, the IRB parameter pointer is set, and the IRB flag is set indicating that the parameter pointer is valid.

10.6.4.2 Initial Processing in the IOU Task.

If the IRB flag indicates that the parameter pointer is valid, IUPRM is called to process the parameter list.

IUPRM is a table-driven module. Each parameter has an associated parameter ID. The parameter ID is used as an index into the table of subroutine entry points. Each subroutine translates a parameter from Name Manager format to the call block format and places the parameter in the correct position of the call block. (See the paragraphs on name management for the format of a parameter list.) For parameters that have no place in the IRB, the parameter (or its address) is saved in the IOU address space for later processing.

10.6.4.3 Channel Operations.

In a Create Channel operation, the channel pathname must be identical to that of the program file containing the owner task, except for the last component. For example, for a program file named .DIRECT.PROGFILE, a valid channel pathname is .DIRECT.CHANNEL. If the SVC call block specifies LUNO 0 as the program file LUNO for the owner task, IOU assumes that the owner task resides on program file .S$SHARED. The SVC returns an error if the owner task is the owner of an existing channel with different scope or if the specified scope is task local and the owner is already the owner of an existing task-local channel.

The SVC causes a CDR to be built in the same directory as the FDR for the program file. The CDR contains the channel description, the installed ID of the owner task, and the record number of the FDR for the program file. The CDR is similar to an ADR and is linked into the chain of ADRs (that is, each CDR or ADR contains the record number of the next CDR or ADR). The CDR can be protected from an accidental Delete Channel operation. The channel access name can be specified in the delete-protect and unprotect I/O operations. When a program file is deleted, its CDRs and ADRs are also deleted. Only the delete-protect flag for the program file is checked when the program file is deleted. See the section on data structure pictures for details of the ADR and CDR.

In a Delete Channel operation, the CDR for the specified channel name is deleted. The program file containing the owner task is not affected.

Channel LDTs have a flag to indicate that they are assigned to a channel. The LDT points to the CCB. The LDT shows the default resource type and type flags carried in the CCB. Each time a LUNO is assigned to a channel, the LUNO count in the CCB is incremented. The count is decremented as LUNOS are released, and the CCB is deleted when the count equals zero.

IOU cannot distinguish an Assign to a channel from an Assign to a file until the CDR is read. After it is read, an FDB and FCB are created for the program file of the owner task. The Assign causes the creation of the CCB and the bidding of the owner task (unless the relevant structures already exist). The CCB contains the eight-character name of the last pathname component for the channel. After an owner task is bid, the run-time ID is used to search the TSB list for the TSB address of the owner task. The calling job JSB address and owner task TSB address are placed in the CCB. Each CCB created by an Assign points to an FDP of the program file containing the owner task. To search for an existing CCB, IOU searches for a CCB with an FMT,FCB pair that matches the desired program file FMT,FCB pair. This indicates that the owner task came from the same program file. The installed ID of the owner task must match that of the owner task of the channel to which the caller is assigning; a match on the last pathname component for the channel is also required.

To establish a global channel, the owner task must be running and issue an Assign to the channel. The CCB is built in the STA, and subsequent Assigns to the channel cause the creation of LDTs that point to this same CCB. Any requester Assigns that precede the owner's Assign will receive errors. The owner of a global channel is identified by its installed ID and the value for TSBFMT and TSBFCB, the description of the program file from which the task was bid.

The first Assign to a job-local channel causes a CCB to be built in the caller's JCA. The owner task is bid by IOU in the caller's job via the SVC option that allows a task to be bid in a different job. Subsequent Assigns to the same channel use the same CCB and owner task.

For task-local channels, a CCB is created and an owner task is bid on each Assign to the channel (excluding Assigns by the owner task). IOU must match up the owner task's Assign with the correct CCB, which was previously created. An error is returned if the owner is the first to assign to a task-local channel.

During initial IOU processing of an Assign to a channel for which the owner processes Assigns, an LDT flag is set to indicate that the LDT is nonusable. The address of the LDT is placed in the

BRO, and a BRO flag is set to indicate that IOU has partially processed this request. IRBSID (session ID) is cleared. An NFMAPO call is issued to the IPC pre-processor to transmit the BRB to the owner task. Parameters associated with this call are passed to the owner task along with the call block.

After the owner task processes the Assign, IPC returns the BRB to the IOU queue. The BRO flag tells IOU that the request has already been processed. The LDT address is obtained from the BRO. If the IRB error code is nonzero, the CCB LUNO count is decremented, and the LDT is released. If no error occurred, the LDT is completed. The resource type is placed in the LDT, and the nonusable flag in the LDT is cleared.

The LDT is built before the owner processes the Assign because the number being assigned must be checked for conflicts before owner task processing. In addition, while the owner is processing the Assign, it must be ensured that no other task assigns the same job-local or global LUNO.

10.6.4.4 Concatenated Files and Multifile Sets.

FCBs for concatenated files are linked together via pointers in the FCBs. The FCBs are flagged as being members of a concatenation, and the first FCB of the set contains the number of files in the concatenation. Concatenated files may be shared as a set. IOU provides error checking to prevent concurrent use of individual files of a set. This provides protection against unanticipated changes in the structure of the concatenated file set. (For example, this prevents another task from changing the end-of-medium on the second of three concatenated files.)

A zero in the first byte of the pathname indicates that multiple pathnames are present. The second byte contains the number of pathnames. Only Name Manager can provide a pathname in this format, because the IOU preprocessor disallows a zero-length pathname. The Name Manager generates a pathname when processing a logical name for the concatenated file.

IOU builds an FCB for each file of a concatenated set. An error is returned if the files are special usage files or if not all of the files are of the same file type. If LUNOS are assigned to any of the individual files, an error is returned. The FCBs are flagged and linked, and the number of files is placed in the first FCB. An RPB is built and linked to the first FCB. Access privileges for concatenated files apply to the set of files, not to each individual file. Open processing checks access privileges for the set of files by searching the RPB chain for the first FCB only.

On an attempt to share a concatenated set, IOU provides error checking. IOU checks to see that the same number of files is specified, and that the requested files are in the same order as those already concatenated. An error is returned if these conditions do not hold. Each usage of the concatenated set causes the creation of an RPB chained to the first FCB. When the concatenated set is no longer in use (when the last RPB is deleted), all FCBs in the set are released.

When key indexed files are combined, they are not considered to be concatenated, but are called a multifile set. This is because after files are concatenated, they can still be used as individual files. This is not true for key indexed files; once a set of files has been combined into a multifile KIF set, they cannot be handled separately using KIF operations.

Multifile KIF sets require special handling. IOU uses the FDR end-of-medium field to determine whether a key indexed file is empty. If all the specified files being combined are empty, and are not members of an existing multifile set, IOU formats them as a single file. The creation date and time of the first file are placed in the KDR of each file in the set. Each file is given a sequence number, which is an integer value that ranges from one to the number of files being combined. The sequence number of each file is also stored in the KDR of the file. The KDR for the first file contains the total number of files in this set.

Error checking is provided when nonempty key indexed files are combined. Each KDR must contain the same creation date and time. Also, the total number of files from the KDR must be the same as the number of files specified in the current combination request. The sequence numbers of the files must start at one and continue sequentially up to the total number indicated in the first KDR. If any of the above conditions is not met, an error is returned. An empty key indexed file may be used as the last (and ONLY the last) file of a nonempty multifile set. The new file must not be a member of an existing multifile set. The new file is formatted and given a sequence number and a time and date to match the other file(s). The file count in the first KDR is incremented.

When a LUNO is assigned to a single key indexed file, the KDR is checked for a sequence number. If one is present, a flag is set in the FCB to indicate that the file can be opened only for unblocked I/O. This is the mode used by the directory utilities, for example.

10.6.4.5 Temporary Files.

DNOS supports two types of temporary files: task and job local. Task temporary files are created either by using the temporary file bit in the Create File operation, or by issuing an Assign LUNO with the temporary file bit set. When the Create File is used, a standard file name can be specified; when an Assign LUNO is used, a name is created in the form #n, where n is a 7-digit integer. Task temporary files are often used as scratch files by system utilities. They are deleted when the last LUNO assigned to the file is released.

Job temporary files are created by specifying the temporary option on an Assign Logical Name operation. The job temporary file is created when an Assign LUNO is done to the logical name or when a Create File specifies the logical name. The access name associated with the logical name is the disk or volume name on which the file is to be created.

IOU creates job temporary files under VCATALOG, and it sets the temporary flag in the FDR. After the file is created, IOU automatically attaches the resource to the job. The file is not detached until the job terminates or the logical name is released. The file is deleted when the count of attaches and LUNOs assigned in the FCB is zero.

After a job temporary file is created, the file name must be entered in the Name Manager data base. For job temporary files, the Name Manager allocates 18 bytes for the pathname (enough for the length, eight-character volume name, period, and eight-character file name). The actual length of the disk or volume name is in the first byte of the pathname buffer. IOU places the length and file name of the newly created file in the buffer. The already-processed flag is set in the BRO, and the BRB is placed on the Name Manager's queue. Name Manager is responsible for calling the routine to queue the BRB for unbuffering.

10.6.5 Operating System Support SVCs.

IOU provides SVC support for several subopcodes that are not available to the general user community. These codes, described in detail in the section on special SVC support, include the following:

Subopcode	Purpose
94	Assign Diagnostic Device
A0	Attach Resource
A1	Detach Resource
A3	Detach Resource by Number
A4	Modify FDR Bit
A5	Release LUNO in Another Job

10.7 DEVICE I/O UTILITY (DIOU)

Device utility functions (bidding tasks from a device; changing device state; disarming hard break at a terminal; allowing 8-bit characters) are performed by issuing device utility operations. These operations are IOU subopcodes and are transportable through

IPC. Since none of these subopcodes conflict with other IOU subopcodes, they are processed by the DIOU task to get concurrency. Some of the subopcodes require software privilege. The operations and their subopcodes are:

Subopcode	Purpose
C2	Get Selected Device Parameters
C3	Set Selected Device Parameters
C6	Get CDE From CDT
C7	Process Device Task Bid

10.7.1 DIOU Functions.

All I/O subopcodes are passed by the SVC request processor RPROOT to IOPREP, the I/O preprocessor. IOPREP hands all I/O subopcodes greater than >90 to IUPREP. IUPREP verifies that DIOU subopcodes are in the proper range. IUPREP buffers the parameter field (if specified) along with the call block in system table area. Control is then transferred to to DUMAIN for specific subopcode processing.

DSRs bid tasks by calling IOFCDT. IOFCDT places the bid character in the PDT and then places the PDT on the BIDREQ queue. When the scheduler finds something on the BIDREQ queue, it activates IOTBID. IOTBID gets the device number and bid character from the PDT and places them in a Process Device Task Bid (>C7) call block. The PDT is removed from the BIDREQ queue, the bid character is cleared in the PDT, and the call is issued to DIOU.

When the CDFPEA flag is set to 1, DIOU passes the bid character in the first byte of the first parameter and leaves the second byte 0. The device number is placed in the second parameter. Logon tasks that need to CDE can then perform a Get CDE from CDT operation (>C6). If the task is to be bid in the same job as PDTJOB, the CDE parameters are placed in the >2B call block parameter fields CDEPV1 and CDEPV2.

10.7.2 DIOU Data Base.

There is one data file for each operating system generated. The file is in the directory S$CDT under VCATALOG. The file within S$CDT has the same name as the system to which it is associated. Each file has 25 CDTs. The file is initialized by the Modify Command Definition Table (MCDT) command using SCI, or by DIOU during IPL if it does not exist.

Each record of the CDT file is a command definition table (CDT). Each record contains the SCI command definition entry (CDE) and hard break CDE, the DXP CDE, and 13 zeroed CDEs. That is, each CDT has a maximum of 16 CDEs. The last four characters of the CDT contains the characters CDT.

Each time a system is loaded, DIOU runs the PDT list and places device numbers into the PDTs.

Each device has a three-byte field in its PDT to represent the CDEs that apply to it. One byte indicates the CDT to use and a word represents the CDEs in the table that are valid for the device. If the first bit is on in the word, the first CDE will be valid for the device; second bit, second CDE; and so on. This way each device may have an unique set of CDEs, with a maximum of sixteen.

10.7.3 Data Structures Used by DIOU.

System data structures used by DIOU include the PDT and a template of parameters. Relevant fields in the PDT include

* PDTNAM - an eight-byte device name field

* PDTNUM - a two-byte device number

* PDTCHR - a one-byte field for the bid character set by IOFCDT

* PDTCDT - a one-byte CDT number

* PDTCDE - the CDE mask for this device's CDT

The structure template for device parameters (DPR) is a set of equates, one for each field in the DIOU data base (multiple flags are stored in one field). Any not marked as read only require either software privilege, hardware privilege or system task status to modify.

```
       UNL
***********************************************************
*                                                         *
*      DUTIL DEVICE PARAMETERS    (DPR)     03/11/83      *
*                                                         *
*      CHANGES TO THIS TEMPLATE REQUIRE CORRESPONDING     *
*      CHANGES TO THE PASCAL TEMPLATE "DPRPAS".           *
***********************************************************
*  THE DPR TEMPLATE DESCRIBES THE DEVICE PARAMETERS MANAGED
*  BY THE DEVICE I/O UTILITY (DUTIL).  IT INCLUDES PARAMETERS
*  IN THE FOLLOWING RANGES:
*
*      PARAMETER RANGE        PARAMETER USAGE
*      ---------------        ---------------
*         >01 - >5F           OPERATING SYSTEM RESERVED
*         >60 - >FF           NOT SUPPORTED
*
*  IN THE FIELD COMMENTS, RO INDICATES THAT A PARAMETER IS
*  READ ONLY AND CANNOT BE MODIFIED.
*
*  SPECIAL FIELD COMMENTS:
*  DPRNAM - ONE TO EIGHT ALPHANUMERIC CHARACTERS WITH A LETTER
*           AS THE FIRST CHARACTER.
*  DPRNUM - ONE WORD NUMBER BETWEEN >0001 AND >07FF, EXCLUDING
*           100 THROUGH 255 (>64 THROUGH >FF).
*  DPRTYP - LIKE THE PDTTYP FIELD.  ON AN ASSIGN LUNO, THE VALUE
*           OF THIS FIELD IS PUT INTO THE LDTTYP FIELD OF THE
*           LDT AND IS RETURNED TO THE CALL BLOCK IN THE UPPER
*           BYTE OF THE DATA BUFFER FIELD.
*  DPRJOB - JSB OF THE FIRST JOB TO ASSIGN A LUNO TO A TERMINAL.
*
DPRNAM EQU  >01     RO   *DEVICE NAME
DPRNUM EQU  >02     RO   *DEVICE NUMBER
DPRFLG EQU  >03          *WORD OF FLAGS
DPRDSF EQU  >04          *DEVICE STATUS
DPRTYP EQU  >05     RO   *DEVICE TYPE
DPRJOB EQU  >06     RO   *OWNER JOB
DPRRPB EQU  >07     RO   *RPB LIST HEADER
DPRLC  EQU  >08     RO   *LUNO COUNT
DPRCDT EQU  >09          *CDT NUMBER
DPRCDE EQU  >0A          *CDE MASK
DPRPDT EQU  >0B     RO   *PDT ADDRESS
DPRDTF EQU  >0C     RO   *DEVICE TYPE FLAGS
DPRSTK EQU  >0D          *SECTORS PER TRACK
DPROHD EQU  >0E          *OVERHEAD PER RECORD
DPRWTK EQU  >0F          *WORDS PER TRACK
DPRDRS EQU  >10          *DEFAULT PHYSICAL RECORD SIZE
DPRFMS EQU  >11          *VCAT FD SPECIAL AREA SSB ADDRESS
DPRFDB EQU  >12          *VCATALOG FDB ADDRESS
DPRTFL EQU  >13          *TEMPORARY FILE NAME SEED
DPRECT EQU  >14          *RETRY COUNT
DPRVNM EQU  >15          *VOLUME NAME
DPRCHR EQU  >16          *BID CHARACTER
```

```
DPRBLN EQU  >17          *BUFFER LEN OR # VIRT TERMINALS
DPRMAX EQU  >18          * THE LIMIT FOR CURRENT OS PARMS
DPROSM EQU  >5F          *MAXIMUM O.S. PARAMETER
* EQUATES FOR DPRFLG
DP1IRB EQU  0       RO   *COPY IRB TO SYSTEM LOG
DP1RS1 EQU  1       RO   *RESERVED
DP1RS2 EQU  2       RO   *RESERVED
DP1STA EQU  3            * DEVICE STATE
*      00 - ONLINE
*      01 - OFFLINE
*      10 - DIAGNOSTIC
*      11 - SPOOLER
DP1OPF EQU  5       RO   *OPEN FAILED
* EQUATE FOR DPRDSF
DP2RS1 EQU  0       RO   *RESERVED
DP2AID EQU  1       RO   *ALTERNATE PDT
DP2BI  EQU  2       RO   *BUFFER INPUT
DP2BO  EQU  3       RO   *BUFFER OUTPUT
DP2JIS EQU  4            *JISCII, 8-BIT ASCII MODE
DP2REN EQU  5       RO   *RE ENTER ME
DP2JAR EQU  6       RO   *JISCII RECEIVE
DP2JAT EQU  7       RO   *JISCII TRANSMIT
DP2RS2 EQU  8       RO   *RESERVED
DP2RS3 EQU  9       RO   *RESERVED
DP2WPM EQU  >A      RO   *WORD PROCESSING MODE
DP2IRE EQU  >B      RO   *INITIAL REQUEST
DP2INT EQU  >C      RO   *DEVICE INTERRUPT LEVEL MASK
* EQAUTES FOR DPRDTF - DEVICE TYPE FLAGS
DP3FIL EQU  0       RO   *FILE ORIENTED
DP3TIL EQU  1       RO   *TILINE DEVICE
DP3TIM EQU  2       RO   *ENABLE TIME-OUT
DP3PRI EQU  3       RO   *PRIVILEDGED DEVICE
DP3KSB EQU  4       RO   *TERMINAL WITH A KSB
DP3COM EQU  5       RO   *COMM DEVICE
DP3SYD EQU  6       RO   *SYSTEM DISC
SP3RES EQU  7       RO   *RESERVED
       LIST
```

DPRNAM

One to eight alphanumeric characters with a letter as the first character.

DPRNUM

A one word number between >0001 and >07FF excluding 100 through 255.

DPRFLG

DPRFLG is a flag word with the following bit definitions.

```
    DP1IRB = (X...............) - COPY IRB TO SYSTEM LOG
    DP1RS1 = (.X..............) - RO   RESERVED
    DP1RS2 = (..X.............) - RO   RESERVED
    DP1STA = (...XX...........) - DEVICE STATE
                                  00 - ONLINE
                                  01 - OFFLINE
                                  10 - DIAGNOSTIC
                                  11 - SPOOLER
    DP1OPF = (.............X..) - OPEN FAILED
    DP1RS3 = (..............XX) - RO
```

DPRDSF

DSPDSF is a flag word with the following bit definitions.

```
    DP2RS1 = (X... .... .... ....) RO  RESERVED
    DP2AID = (.X.. .... .... ....) RO  ALTERNATE PDT
    DP2BI  = (..X. .... .... ....) RO  BUFFER INPUT
    DP2BO  = (...X .... .... ....) RO  BUFFER OUTPUT
    DP2JIS = (.... X... .... ....) RO  JISCII, 8-BIT ASCII MODE
    DP2REN = (.... .X.. .... ....) RO  RE ENTER ME
    DP2JAR = (.... ..X. .... ....) RO  JISCII RECEIVE
    DP2JAT = (.... ...X .... ....) RO  JISCII TRANSMIT
    DP2RS2 = (.... .... X... ....) RO  RESERVED
    DP2RS3 = (.... .... .X.. ....) RO  RESERVED
    DP2WPM = (.... .... ..X. ....) RO  WORD PROCESSING MODE
    DP2IRE = (.... .... ...X ....) RO  INITIAL REQUEST
    DP2INT = (.... .... .... XXXX) RO  DEVICE INTERRUPT LEVEL MASK
```

DPRTYP

The DPRTYP field corresponds to the PDTTYP field. On an Assign LUNO, the value of this field is placed in the LDTTYP field of the LDT as well as being returned in the upper byte of the data buffer field of the Assign LUNO call block.

DPRJOB

DPRJOB is the same as PDTJOB. It is only valid for devices with a KSB (terminals). The JSB of the first JOB to assign a LUNO to terminal is placed in DPRJOB. This prevents any other JOB from assigning a LUNO to the terminal.

DPRRPB

DPRRPB is the same as PDTRPB. For devices that validate opens an RPB must be generated during assign LUNO processing. DPRRPB contains the address of the beginning of the RPB list.

DPRLC

DPRLC is the same as PDTLC. This field contains a count of the number of LUNOs assigned to the device.

DPRCDT

The first byte of DPRCDT identifies the CDT for this terminal. The next two bytes are the word mask that identifies the CDEs within the CDT that are valid for this terminal.

10.8 FILE ACCESS SECURITY

DNOS 1.2 includes security on a per file basis. Directory security and volume security are not implemented in this release.

A user's security environment is defined at the job level. Associated with each job in the system is a user ID and the list of access groups of which he is a member. A collection of users which are expected to have similar access to files belong to an access group. All secured files have a list of access groups and rights which determine who may access the file and in what manner. This list is called an access control list. Permission is granted if the user is a member of an access group in the access control list.

Since security is enforced on a per job basis, it is not possible for a server job to accept files from other jobs and perform operations on those files with the original user's security. The spooler is an example of such a server job. To solve this problem, an option for secured IOU SVCs is supported. A task can specify the user ID with which the access rights for the LDT will be generated. Such a task must have the security bypass feature and enforce its own security, unless the passcode for the user ID is also specified. Specific operations that have this feature include Assign LUNO, Create File, Delete File, Modify File Name, and the Modify File Protection SVCs.

Assigning a LUNO to the disk (using direct disk I/O) and bidding a task in another job are potential security bypass operations that are not restricted by the operating system. These are better handled through the use of functional security or by allowing tasks to be members of access groups, features that may be implemented in a future release.

10.8.1 Establishing a Job's Security Environment.

A user's security environment is determined during job creation. The logon task solicits a user ID and passcode. If logon is turned off through use of the MTS command, this information is located in the S$SCA file. The user ID and passcode are validated against the entries in the S$CLF file.

Job Manager, during create job processing, validates the user ID and passcode against the S$CLF. This check is necessary since not all create jobs are issued by the logon task. Job manager reads the S$CLF to find the user ID, passcode and access groups. The user ID is copied into the JSB. The encoded passcode is copied into the JIT. A capability list is built. This list includes a count of access groups followed by a list of encrypted access group names, each name followed by a flag word. This list resides in the JCA, pointed to by JITCAP.

The security manager function is performed by an overlay of IOU. This overlay is not included as part of the kernel program file if the security option is not chosen during sysgen. In general, a user's security access to a file is determined during assign LUNO processing. The rights are stored in the LDT. Subsequent operations to the file check the rights in the LDT to determine whether the user has appropriate access.

All IOU operations are first processed by IUMAIN. IUMAIN maintains a table of secured IOU operations. These operations include Create File, Assign LUNO, Delete File, Modify File Name, Unprotect File, Write Protect File and Delete Protect File. Several conditions must be met before security checking is done: the system must be secured, one of the above secured subopcodes must have been encountered, and the pathname must be longer than a single node (indicating possibly a file as opposed to a device).

IUSPRE is the security manager preprocessor routine. If a user ID parameter is passed with the IRB, IUSPRE verifies that it is valid. If the parameter was not specified by a security bypass task, the passcode is also verified. A capabilities list is generated from the S$CLF file instead of using the capabilities list pointed to by JITCAP.

For all secured suboperations except Create File, IUSPRE calls IUGAR to generate the user's access rights to the file. IUGAR calls the IOU routines that build the FDB tree and FCB for the file. It calls a routine to compare the capabilities list with the file's access control list. If the access group in the capabilities list is SYSMGR, the user receives total access to the file. If no access groups are associated with the job the user will receive PUBLIC access to the file. If no access groups are found and no PUBLIC rights are specified, the operation will fail. The user will be returned a security violation error. If some access is determined, control is returned to IUSPRE.

If the operation is an Assign LUNO, the rights are saved off to be later entered into the LDT during the Assign LUNO routines. If the operation is a Delete File or a Modify Protection SVC, the proper access is verified. In the case of a Modify File Name, the access is verified for the new pathname, if it existed. The access for the old pathname is verified by looking at the LDT rights field. A Modify File Name of a directory is specifically not allowed.

A Modify File Name option exists which allows the user to keep the security of the new pathname. If this option is specified, the security is copied to system table area in a structure known as an access control packet. After the Modify File Name takes place, the security is copied from the access control packet to the file and the packet is released. If the option is not chosen, the default results of the Modify File Name leave the file secured with the old pathname's access rights.

If a LUNO is being assigned to a concatenated file, the concatenated file processor calls the security routines to determine the access rights for each file. These rights are ANDed to determine the final access rights.

In the case of file creation or Assign LUNO with auto-creation of a file, IUSPRE calls a routine to search the capability list to find the creation access group. If no such group is specified, the default is taken as PUBLIC. When the file is created, the encrypted group name is written to the FDR with all access right flags set, and the public bits are turned off.

Aliases inherit the security of the file to which they refer. The addition or deletion of aliases is not secured.

Minimal channel security is implemented. IUSPRE saves the rights to the channel owner program file. Before a channel owner is auto bid by an assign LUNO to the channel, the rights are checked for execute access.

10.8.2 Enforcing Security.

The security manager does some set up of the security environment and enforces security for IOU operations. Security is enforced in each process that performs a protected or a particularly dangerous function.

10.8.2.1 File Manager.

There are differences between open access privileges and security rights. The access privileges specified on an Open operation enable a user to limit access by others while performing a file operation. It may be desirable for a user who has only read access rights to a file, for example, to open it with exclusive use. That is, he may only read the file, but while doing so, no operation to this file by others may take place.

File manager enforces read and write security to files. The file manager routine IOPREP compares the I/O subopcode against a table of allowable operations. The tables and required access are given below.

File operations allowed only if the requestor has read access:

>09 - Read ASCII
>0A - Read Direct
>41 - Read Greater
>42 - Read by Key, Read Current
>44 - Read Greater Than
>45 - Read Next
>48 - Read Previous
>59 - Multiple Record Read

File operations allowed only if the requestor has write access:

>02 - Close with EOF
>0B - Write ASCII
>0C - Write Direct
>0D - Write logical EOF
>10 - Rewrite
>46 - Insert
>47 - Rewrite
>49 - Delete by Key
>5B - Multiple Record Write

File operations which will succeed if Assign LUNO was successful:

Open
Set Currency
Forward, Backward Space LUNO
Read File Characteristics
Unlock
Close
Specify Write Mode
Rewind

10.8.2.2 Program Manager.

Program management enforces security on task bids and overlay loads. NFTBID checks the program file LDT for execute access when bidding tasks. Overlay loads are checked by PMOVYL. Task installations, deletions and assigning program space are enforced by file manager when the user tries to read or write to the program file. S$SHARED is assumed to be a public program file and cannot be secured. When LUNO >FF is specified, no security check is made since execute access to the program file has already been established.

10.8.2.3 Segment Manager.

In order to do a Change Segment operation for a program file segment, the user must have execute access to the program file. In order to do relative record I/O using a Change Segment SVC, the user must have read and write access to the relative record file. In order to do relative record I/O using a Create Segment SVC, the user must have write access to the relative record file. The above is enforced by segment management in SMPREP.

10.8.2.4 Sysgen.

In DNOS 1.1, if the SVC for security was specified the Encrypt and Decrypt SVCs were included. In DNOS 1.2, the ENCRYPTION SVC group includes those SVCs. In DNOS 1.2, a YES response to the global SECURITY question causes the system option word security bit in NFDATA is to be set, the security overlay for IOU to be added to the kernel program file, and the encryption SVC group to be included.

10.8.3 Volume Security.

A utility called Modify Volume Security (MVS) is supplied to make it impossible to install a secured DNOS volume on a DX10 or an unsecured DNOS system. It is located in the .S$SECURE program file. This task modifies a field in sector zero which indicates to the install volume processor that this can only be installed on a secure system.

10.8.4 Networking.

For DNOS 1.2, the security access one will receive doing remote I/O will be the access of the remote LAN job, as the SVCs are issued remotely by this job. When performing a remote logon, one acquires the security associated with the user ID used to log on.

10.8.4.1 Manipulation of the Access Control List.

Modifications to the access control list for a file are made by a system task called MSAR. This utility task allows a user to modify the security access rights of any file for which the user has the control access right. It also allows a user to display the list of access groups which currently have access to the file and what access rights each group has to the file.

MSAR is bid by the MSAR and LSAR SCI command procedures. The PARMS list is as follows:

PARM	Definition
1	Function code: 0=LSAR; 1=MSAR
2	User's passcode
3	File pathname
4	Listing access name (not used for MSAR)
5	Access group name (not used for LSAR)
6	Read access (YES/NO) (not used for LSAR)
7	Write access (YES/NO) (not used for LSAR)
8	Delete access (YES/NO) (not used for LSAR)
9	Execute access (YES/NO) (not used for LSAR)
10	Control access (YES/NO) (not used for LSAR)

The MSAR task consists of the modules MSMAIN, MSMSAR, MSLSAR, MSDOOR, MSDDIO, MSFUDR, and MSRCLF from the VOLOBJ.MSAR.OBJECT directory, several low level subroutines from the VOLOBJ.IOU.OBJECT directory, the passcode encryption subroutine from the VOLOBJ.SECURITY.OBJECT directory, and stardard UTCOMN and S$ routines.

MSMAIN verifies that the utility is being run on a DNOS 1.2 (or later) system. It then assigns and opens a LUNO to .S$CLF, picks up the user's ID from the JSB, finds the user descriptor record (UDR) within .S$CLF for the user, and verifies the passcode entered by the user. If the passcode verifies, MSMAIN then assigns a LUNO to the user specified file, finds the LDT for the LUNO just assigned, and verifies that the file is a local file, not a directory, and that the user has the control access right for the file. If all goes well, MSMAIN then calls IUGFCB to map in the FCB for the file and saves the FCB address for later direct disk I/O operations. MSMAIN then goes to either MSLSAR or MSMSAR depending on the operation specified.

MSLSAR closes and releases the LUNO to .S$CLF as the utility is done with that file. It then calls S$OPNS to open the listing file with the security rights of the requesting user ID. This special entry point in the S$OPEN routine must be used because the MSAR task is installed with the security bypass attribute, but the user must not be allowed to put the listing on top of files to which he does not have write access. MSLSAR then calls MSRFDR to read the FDR for the user specified file and formats the listing to show the various access groups that have access to the file and the access rights for each group.

MSMSAR calls S$PARM to get the user specified access group name. If the name is SYSMGR, MSMSAR terminates with an error message. Otherwise, MSMSAR verifies that the access group name is valid on this system. If the name is PUBLIC, then it is considered valid. Otherwise, MSMSAR calls MSRCLF to read records from .S$CLF and verifies that the name is a valid existing access group name. At this point, MSMSAR closes and releases the LUNO to .S$CLF since the utility is now done with that file. MSMSAR then calls S$PARM several times to determine the various access rights which the specified for the access group to have. MSMSAR then calls MSDCLO to close the door on the directory in which the FDR for the user specified file resides, calls MSRFDR to read the FDR for the file, changes the access control list appropriately, calls MSWFDR to write the FDR back to disk, and then calls MSDOPN to open the door for the directory. MSMSAR then releases the LUNO to the user specified file and terminates.

10.9 INTERPROCESS COMMUNICATION (IPC)

IPC provides the capability for two or more tasks to exchange information. The information exchanged may take the form of a synchronization signal, a short message, a request for a service, or a high-volume data transfer. The distinction between these categories is often blurred in practice.

The primary IPC operations are to read and write messages. The task to which a write or read is addressed may be identified in several ways. In a message-oriented IPC mechanism, no permanent channel exists between the two communicating processes. Writes and reads are addressed to specifically named processes, and the IPC supervisor must utilize a rendezvous table to resolve matching operations. The IPC approach in DNOS is channel oriented. Channels are created and exist independently of the tasks that use them; IPC requests are directed to the appropriate channels.

In DNOS, a channel is defined as an IPC path between two different tasks within the same computer, with one of the tasks designated as the owner of the channel. A channel owner has control over how the channel is used, while the second task (the requester) has less flexibility and fewer privileges.

IPC processing in DNOS has the following characteristics:

* Each write or read operation is addressed to a channel, to which a LUNO has been assigned.

* Each operation issued to a channel by a requester must be matched by an operation of the channel owner. Depending on the operation issued and the current environment, the requester may or may not be suspended until the operation completes.

* Each channel has a queue of pending requests to the owner, ordered chronologically.

IPC channels may be used via SCI like other I/O resources on a DNOS system. Some of the SCI commands available for use with IPC channels are the Create IPC Channel (CIC), Delete IPC Channel (DIC), and Show Channel Status (SCS) commands.

10.9.1 IPC SVC Interface.

All program-level access to the IPC facility is through the DNOS SVC mechanism. The general SVC parameter block used for IPC service calls to symmetric channels is like that used for resource-independent I/O. The SVC parameter block used for request SVCs to master-slave channels may be like that used for resource-specific I/O. The parameter block for utility calls is like that used for file and I/O utility calls. The IPC opcodes are shared with the DNOS I/O facility, with the operation performed depending on the context.

Every channel has one owner. The owner of a channel is always involved in every message exchange or other operation on that channel. Any user other than the owner of the channel may communicate only with the owner. This results from the matching rules used by IPC when handling requests to channels. These rules are given in the description of the IPC support routine, IPCGQR.

10.9.2 Channel Characteristics.

DNOS channels can be defined as either symmetric or master/slave. Symmetric channels function with simple read and write requests, where one correspondent on the channel issues a read while another issues a write. The data written is passed 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 task processes and returns the modified request to the requester (slave) task.

10.9.2.1 Symmetric Channel Activity.

Symmetric channels are used for communicating messages or data in a relatively restricted fashion. Tasks may be written in high-level languages or in assembly language to synchronize, exchange information, or facilitate use of common files. The operations addressed to the channel by such tasks are limited to Open, Close, Close EOF, Read Status, Read, Write, and Write EOF.

From the point of view of an owner or requester task, a symmetric channel is always in one of three states:

* Closed - The task must issue an Open to use channel.

* Open - The task may issue any operation that is legal in this channel access mode.

* Dormant - The task must issue a Close and then may again open and use the channel.

The transitions between states are shown in Figure 10-14.

An Open to a channel, when fielded by the I/O preprocessor, is checked against current access privileges held by other requesters. The same rules apply for channels as for other I/O resources when granting or denying access to any user after the first. For example, if one requester opens a channel with privileges of exclusive all, no other requester can open the channel.

Most symmetric channel requesters issue Opens with shared access. When the requester wishes to be the only requester served, the channel should be created as a non-shared channel.

IPC uses the access privileges specified on an Open to allow or deny particular I/O operations to a symmetric channel. For example, a user who opens with read only privileges might issue a Write. That Write is not allowed (just as it is not allowed to a read-only device like a card reader).

Although requesters may not be aware of the fact, the channel definition may dictate more stringent access privileges than they specify. For example, a symmetric channel established as a nonshared channel can be opened by a requester in any mode, but it will function with only one requester at a time. This is necessary for the read/write channel resource mode of a symmetric channel, since the owner task has no way of differentiating between requesters.

A Close may be issued by an owner or a requester, or it may be issued by DNOS when processing an abnormal termination by a task on the channel. IPC adjusts the CCB to reflect the Close; the I/O postprocessor modifies the LDT as it does for devices. Any queued requests from the closing task are removed from CCB queues.

When a Close is issued by an owner, IPC fields the request and the channel is marked as dormant for all requesters currently open. This setting causes requesters to receive errors on any operations except Open and Close. As soon as a requester on a dormant channel issues a Close, the channel is again closed for that requester; however, it is available to be opened. Opens to a dormant channel are queued to the CCB until each Close has been processed and the channel owner can again issue an Open.

A Read Device Status operation is queued to IPC for processing as soon as the IPC task executes.

```
                          +---------------+ req. open
                          |CLOSED to      |---------+
                          |     owner     |         |
                          |CLOSED to      |         |
                          |     requester |         |
                          |     OOP=NO    | <-------+
                          |     OCL=NO    |    req. hangs
                          |     RCL=NO    |
                req.+---->|     OPN=0     |<---+
               close|     +---------------+    |
                    |                   |      |owner close
 +----------+       |     owner|        |owner |      +--------+
req.    req. gets   |     close|        |open  |    owner    owner
non-    >A7 error   |          |        |      |    non-    gets >E6
close       |       |          |        |      |    close    error
 |          |       |          |        |      |      |        |
 |          V       |          |        V      |      |        V
+------------------+    +-----------------+    +-----------------+
|CLOSED to         |    |OPEN to          |    |DORMANT to       |
|     owner        |    |     owner       |    |     owner       |
|DORMANT to        |    |CLOSED to        |    |CLOSED to        |
|     requester    |    |     requester   |    |     requester   |
|     OOP=NO       |    |     OOP=YES     |    |     OOP=YES     |
|     OCL=YES      |    |     OCL=NO      |    |     OCL=NO      |
|     RCL=NO       |    |     RCL=NO      |    |     RCL=YES     |
|     OPN=1        |    |     OPN=1       |    |     OPN=1       |
+------------------+    +-----------------+    +-----------------+
 |owner open                  |                                   |
 |(owner     |     |         |            |req.    |      |req.  |
 |  hangs)   |     |         |req.        |close   |      |open  |
 +----->-----+     |         |open        |(shared |      +--<---+
                   |         |            |channel)|        (hang)
              owner|         V            |        |
              close|      +------------------+     |req. close
                   +---<--|OPEN to owner     |-->--+(non-shared
                          |OPEN to requester |       channel)
                          |     OOP=YES      |
                          |     OCL=NO       |
OOP - Owner Open          |     RCL=NO       |
OCL - Owner Closed        |     OPN=2        |
RCL - Requester Closed    +------------------+
OPN - Number of tasks                        |All operations
      open                           |       |excluding opens
                                     +----<--+and closes
```

Figure 10-12 Symmetric Channel States

On other operations to a symmetric channel, a match must be found by IPC before the operation will be performed. That is, one task must issue a Read and the other a Write before either operation will be processed.

The owner task may have only one request outstanding to a particular channel at any one time. If an owner issues a request to a channel to which an owner request is already pending, the second request will be returned with an error.

10.9.2.2 Master/Slave Channel Activity.

A master/slave channel is established so that an owner task may process all requests of the requester tasks. The master/slave owner task can be written in assembly language using the Master Read, Master Write, Redirect Assign Luno, and Read Call Block operations. It may be written in a high-level language with subroutine support to issue these SVCs.

When accessing a master/slave channel, a requester may need to pass a set of parameters to the master (owner) task. These parameters may be specified as part of an Assign Logical Name command and passed to the owner task via an Assign LUNO operation.

In this and other cases an owner task may process requester Assign LUNO and Release LUNO operations, gathering and supplying data from and to IOU. This option is specified by the Create IPC Channel SVC operation or SCI command. The owner receives the request with Master Read and modifies it to reflect appropriate processing. The owner task then issues a Master Write of the Assign LUNO call block. IPC queues the request back to IOU to be completed. The owner task must not modify certain fields in the SVC block, so that IPC can correctly return the block.

Similarly, the owner may process Abort I/O SVCs (>0F) or I/O utility operations other than Assign LUNO and Release LUNO. These options can be specified by the Create IPC Channel SVC operation or SCI command.

While using a master/slave channel, a requester task may issue any I/O operation to a channel, and the owner task processes the operation depending on how the owner task is designed. The owner task issues a Master Read to obtain a request for processing and a matching Master Write to return messages or status information to the requester.

Owners of master-slave channels may have only one request outstanding to a particular channel at any one time. The Master Write and Redirect Assign LUNO operations are exceptions to this rule. An owner may issue either a Master Write or a Redirect Assign LUNO while a Master Read, Read Call Block, or Read Status operation is pending.

All I/O operations issued by the requester are passed to the owner task by IPC. Figure 10-15 shows the operations used by the owner task.

00	Open	IPC executes the Open checks for legal access
01	Close	IPC executes the Close
05	Read Status	Owner receives status data
19	Master Read	Owner gets whole call block of requester
1A	Read Call Block	Owner gets first part of requester call block
1B	Master Write	Owner sends information to requester
1C	Redirect Assign LUNO	IPC sends call block to another channel owner

Figure 10-13 Owner SVCs for Master/Slave Channels

10.9.3 Details of IPC Processing.

Service calls to IPC channels are first routed through the I/O preprocessor, IOPREP, for common I/O handling. They are then passed on to the IPC preprocessor, IPCPRE. This routine checks for some error conditions. If no error is found, IPCPRE determines whether fast transfer is possible. If fast transfer is possible, the exchange is performed immediately. Otherwise, the request is queued to the IPC queue server, IPCTSK, and the requester task is suspended.

10.9.3.1 Structures Used for IPC Processing.

Each channel is represented by a channel control block (CCB) in the system table area or the job communication area. The CCB contains two queue headers: the pending queue (CCBPBQ) and the already-processed queue (CCBABQ). All requests awaiting processing by IPC other than Master Write and Redirect Assign LUNO requests are placed on the CCBPBQ. Master Write and Redirect Assign LUNO requests are placed on the CCBABQ. Requester call blocks which have been master read but have not yet been master written or redirected are stored on the CCBABQ. There is never more than one owner request on any queue. If there is an owner request on a queue, it will always be the first entry on the queue.

The IPCQUE, which is in the system table area, contains requests for task level IPC processing. The queued requests are processed by the IPC queue server IPCTSK. The queue entry is a QIR, shown in the section on data structure pictures. The queue link field is used to chain all current requests to be performed by IPCTSK. If the channel being used for this queue entry is global, the JSB address (to find the CCB) is zero. This indicates that the CCB and BRBs are in the STA. For other types of channels, the JSB is used to find the segment information so that the JCA segment can be loaded into memory and the CCB and BRBs can be located.

The anchor for IPCQUE is in the STA. The anchor is a standard DNOS queue header, shown by the QHR template in the section on data structure pictures. When NFQUEH places an entry on the IPC queue, IPCTSK is bid. IPCTSK processes each queue request by examining the CCBABQ and CCBPBQ for the CCB specified in the QIR and matching as many requests as possible.

10.9.4 Detailed Operation of IPC Routines.

There are two paths through the IPC subsystem: a task level path and an XOP level (fast transfer) path. All IPC requests are handled by the IPC preprocessor, IPCPRE. IPCPRE determines whether a request can be handled in fast transfer mode. If so, IPCXFR is called to transfer the request to the IPC task (which must be in memory). The IPC task, running in XOP mode (map 0), processes the request. The routine IPCXOP in the IPC task handles the request. If IPCPRE determines that fast transfer is not possible, the request is queued and later processed by the IPC queue server, IPCTSK. The same code (IPCPRO, IPCMRD and IPCMWT) is used to perform both XOP level and task level IPC processing. Fast transfer is only possible when the IPC task and all necessary buffer segments are in memory.

If a channel owner releases its LUNO to the channel, or if the owner of a task-local or job-local channel terminates, the channel is considered dead. This is indicated by the CCB flag CCFDED. IOU or PMTERM will set the CCFDED flag, build a QIR for that channel, and queue it to IPCQUE in order to activate IPC. IPC will return outstanding and subsequent requester requests with an owner aborted error, (error code >A7).

10.9.4.1 IPC Preprocessor (IPCPRE).

IPCPRE runs at XOP level when an I/O or IOU call buffered by IOPREP or IUPREP is detected as being for a channel or remote resource. The following is a description of the IPCPRE algorithm:

```
IF the request is an I/O request or an Abort I/O request
THEN BEGIN
     IF the request requires a buffer
     THEN call IPCCB to build a Buffer Address Packet (BAP)
          for the buffer.
     IF the channel is not busy and the IPC task is in memory and the
               request is not a Redirect
     THEN BEGIN
          call IPCXFR to process request in fast transfer mode.
          IF IPCXFR returns without error
          THEN return.
*             If IPCXFR returns an error, one of the task
*             segments or buffer segments was not in memory,
*             so fast transfer was not possible.  The request
*             has been queued by IPCXOP to the CCB.  All we
*             have to do now is queue a QIR to the end of IPCQUE
          END
     ELSE IF request is a Master Write or Redirect Assign LUNO
          THEN queue request to head of CCBABQ.
          ELSE IF request is an owner request
               THEN queue request to head of CCBPBQ.
               ELSE queue request to end of CCBPBQ.
     END
generate a Queued IPC Request (QIR).
queue the QIR to the end of IPCQUE.
set the Channel Busy flag in the CCB.
     return.
```

10.9.4.2 IPC Queue Server (IPCTSK).

When a request cannot be processed in XOP mode, the request is processed at the task level by IPCTSK. IPCTSK is activated by NFQUEH when a QIR is queued to IPCQUE. The following is a description of the IPCTSK algorithm.

```
WHILE the IPCQUE is not empty
     BEGIN
     get request from IPCQUE.
     IF channel is job or task local
     THEN map the JCA that contains the CCB
     WHILE there exist processable requests on the
              CCBPBQ or CCBABQ
          BEGIN
          call IPCGQR to get a request or a pair of
                   requests.
              IF IPCGQR returned an owner request
              THEN IF the owner request requires a buffer
          THEN load the buffer segment.
              IF IPCGQR returned a requester request
              THEN IF the requester request requires a buffer
          THEN load the buffer segment.
              IF this is a master/slave channel and the owner
                   request is not a Redirect or a Read Call Block
              THEN load all requester task segments.
              call IPCPRO to execute the data exchange.
          IF owner request was not processed
              THEN requeue the request to the CCB.
          IF requester request was not processed
              THEN requeue the request to the CCB.
              unload any task segments that were loaded.
     END
     reset the Channel Busy flag in the CCB.
     END
```

10.9.4.3 IPC XOP level request processor (IPCXOP).

The routine IPCXOP is in the IPC task, although it is only executed in map 0. If IPCPRE determines that the IPC task is in memory, a branch and link is performed to IPCXFR, which transfers control to IPCXOP.

IPCXOP queues the request to the CCB and calls IPCGQR to get a request or pair of requests from the CCBABQ or CCBPBQ. The subroutine IPCCHK is called to determine whether the requests returned by IPCGQR can be processed in fast transfer mode. If a request cannot be processed, either because of an error or because fast transfer is not possible, the request is requeued for processing by IPCTSK.

IPCXOP continues to process requests from the CCBPBQ and CCBABQ until a request cannot be processed.

10.9.4.4 IPC request processors (IPCPRO, IPCMRD and IPCMWT).

IPCPRO, IPCMRD, and IPCMWT are the routines in the IPC task that actually perform the data exchange between requests. IPCPRO is called by IPCTSK or IPCXOP for all requests. IPCPRO calls IPCMRD and IPCMWT to process Master Reads, Master Writes and Redirect Assign LUNO operations. Other operations to master/slave channels and all operations to symmetric channels are performed directly by IPCPRO.

On a Master Read or Read Call Block, IPCMRD or IPCPRO copies the requester's call block into the master task's master read buffer (MRB). On a Master Read, the call is dequeued from the CCBPBQ and placed on the CCBABQ to await a Master Write or a Redirect Assign LUNO. On a Read Call block operation, the requester call block is left on the CCBPBQ. On a Master Write, IPCMWT unbuffers selected fields of the requester call block from the copy in the MRB into the copy buffered in system table area. Assign LUNO and Release LUNO call blocks are queued for reprocessing by IOU. End of record processing is performed on all other call blocks.

The initial processing of the Redirect Assign LUNO operation is similar to that of a Master Write of an Assign LUNO call block. Selected fields of the call block are unbuffered from the MRB into the copy of the call block in system table area. Additional processing required for Redirect Assign LUNO is as follows:

1. The LDT that was built by IOU while it was processing the requester call block must be deleted. IPC accomplishes this by obtaining some table area, building a call block for an >A5 I/O operation, and queueing it to IOU. An >A5 is a Release Luno in Another Job SVC. It is documented in the section on special SVCs. The call block must look to IOU as if it has already been processed once by IOU (otherwise, IOU will note that the channel processes assigns and releases and will give the call block back to IPC). IOU normally converts an >A5 into a >93 (Release Luno) by changing the subopcode to >93, setting the >A5 flag (BRFA5) and the Already Seen flag (BRFARS) in the BRO flags, and swapping the JSB and TSB in the >A5 call block with the BROJSB and BROTSB. IPC must duplicate all of these actions. When IOU processes the call block, the LDT will be deleted.

2. The pathname in the MRB must be buffered into the requester call block in system table area. IPC does a Get Table Area SVC, copies the pathname into the table area buffer, and changes the pathname pointer in the requester call block (IRBPNA) to point to the new pathname. This pathname specifies the channel to which the Assign LUNO is to be redirected. The field IRBRPN (redirected resolved pathname), which is a null pointer for all call blocks that have not been redirected, is set to the previous value of IRBPNA. Subsequent

redirects of this call block will not alter the IRBRPN field. The IRBRPN field points to the resolved original pathname.

3. The Already Seen flag (BRFARS) in the BRO flags is reset so that the Assign LUNO call block appears not to have been processed by IOU. The call block is then queued to Name Manager. Name Manager will queue the call block to IOU, and IOU will build a new LDT for this Assign LUNO call block.

4. End of record processing is done on the Redirect Assign LUNO call block.

10.9.4.5 IPC Support Routines.

IPCEOR

IPCEOR performs end of record processing on requests. The BAP is released and any buffer segments are unreserved. If the processing is being done in map 1, IPCEOR calls NFEOR to perform end of record processing. If in map 0, IPCEOR calls IPCOBR to complete the end of record processing.

IPCOBR (in module IPCXFR)

IPCOBR loads the scheduler map file, calls NFEOBR to perform end of record processing, and then reloads the IPC map file.

IPCCB

IPCCB creates a BAP, which is described in the section on data structure pictures. The BAP describes the data buffer of an IPC request.

IPCXFR

IPCXFR transfers control between the SVCSHD segment and the IPC code segment in map 0, both of which must be mapped as the third segment. The IPC map file is loaded and IPCXOP is called. After IPCXOP returns, the original map file is reloaded and control returns to IPCPRE.

IPCGQR

IPCGQR dequeues entries from an IPC channel for processing by IPC opcode processors using the following algorithm:

```
WHILE there are processable requests on the CCBABQ or CCBPBQ
     BEGIN
     IF there is an owner request on the CCBABQ
     THEN BEGIN
          dequeue the owner request.
          get the MRBSSI and MRBRCB fields from the call
                    block in the MRB.
          IF there is a requester PRB on the CCBABQ
               at the address specified by the MRBSSI and the
               BRORCB field equals the MRBRCB field.
          THEN BEGIN
               dequeue the requester call block from the CCBABQ.
               RETURN to the calling task, returning the matched
                   pair of requests.
               END
          ELSE do end-of-record processing on the Master Write,
            returning a NO MATCHING REQUEST error.
          END
     ELSE IF the channel is dead (CCFDED is set) and there is
                 a request on the CCBABQ
          THEN dequeue and RETURN the request.
     ELSE IF there is a request on the CCBPBQ
          THEN BEGIN
               IF the request is not from an owner
               THEN IF the request does not require a matching
                       owner request OR IF the channel is dead
                    THEN dequeue and RETURN the request.
                    ELSE RETURN no requests (no match is
                         available since owner requests always
                         precede requester requests on the
                         queue)
          ELSE BEGIN
               dequeue the request
               IF the owner request does not require a
                  matching request
               THEN RETURN the request
               ELSE IF there is a requester request on
                       the CCBPBQ
                    THEN IF the requester request requires a
                            match
                         THEN dequeue the requester request
                              and RETURN both requests.
                         ELSE BEGIN
                              requeue the owner request.
                              RETURN the requester request.
                              END
                    ELSE requeue the owner request
     END
```

IPCOPY

IPCOPY is used by the IPC request processing routine to copy data buffers. IPCOPY uses the nucleus routines NFXCPY and NFCOPY to perform the data transfer. IPCOPY expects a source address, a destination address, and the number of bytes to transfer. The addresses are either specified as BAPs (for long distance data transfer) or local addresses.

10.10 NAME MANAGEMENT

The Name Manager task handles synonym and logical name segments for jobs running under DNOS. It serves SVC requests from user tasks and supplies support functions to various pieces of the I/O subsystem and to task management. The form of the Name Management SVC block is shown in the section on data structure pictures as the NRB.

Each DNOS task operates with a set of synonyms and/or logical names which is known as a stage. A stage descriptor block (SDB) is maintained in the synonym and logical name segment to describe the stage and its relationship with other stages within a job. Stages are maintained in a hierarchical order. When a daughter stage is created, it is given a snapshot of its parent's names. This is implemented logically rather than making physical copies. The stage under which a task is executing when it issues an SVC is known as the current stage of the task.

The Name Manager serves a queue of requests that include user-issued SVCs for SVC opcode >43 and task management entries to show when a task is bid or terminated. Also, IOU queues BRBs for the following I/O SVCs:

Subopcode	Description
90	Create File
91	Assign LUNO
92	Delete File
94	Assign Diagnostic Device
95	Rename File
96	Unprotect File
97	Write Protect File
98	Delete Protect File
99	Verify Pathname
9A	Add Alias
9B	Delete Alias
9D	Create IPC Channel
9E	Delete IPC Channel
A0	Attach Resource
A1	Detach Resource
A4	Modify FDR Bit

Several of the Name Management SVC subopcodes are described in the DNOS SVC Reference Manual, since they are useful in user-written code. The subopcodes not described in that manual are described in the section on special SVCs in this manual.

For IOU requests, the Name Manager resolves logical names and then places the BRBs on the IOU queue along with any applicable information, such as parameters. These entries may be placed back on the Name Manager queue by IOU if a pathname for a job temporary file has been autogenerated.

10.10.1 Architecture of the Name Manager.

The Name Manager consists of a preprocessor that works in conjunction with the SVC request processor to completely buffer the request, a queue server task, and a special postprocessor that works in conjunction with the main SVC unbuffering routine to return information into the user's address space.

10.10.2 Data Structures Used by the Name Manager.

Each name segment used by Name Manager uses several data structures. Names are organized in stages, with each stage representing a complete environment of names. Each stage is described by a Stage Definition Block (SDB), including a pointer to the stage from which it was built, a task count of the number of tasks using the stage, and a pointer to the descendent error list.

Names and their values are organized in balanced binary trees of Name Definition Blocks (NDB). An NDB has the name, pointers to the left and right son, a pointer to the lexical successor, a pointer to the parent name, and a pointer to the stage value block (SVB) list.

The SVB has flags, a stage number, a pointer to a value definition block (VDB) and a pointer to the next SVB. The SVBs are kept in descending numerical order of stage number. The VDB has a value of a name, a count of the number of SVBs with this value, and a pointer to a value continuation block (VCB). The VCB has a value and a pointer to another VCB. A VCB is used when a name has concatenated files or parameters as values. Figure 10-14 shows the relationship of these various structures for a segment with three names: A, B, and C. Name A is used in three stages and has the value .X.Y.FILE.

```
                   NDS
                +--------+
        +------>|   B    |<-------+
        |       +--------+        |
NDS     V                    NDS  V
    +-------+                  +-----+
    |   A   |                  |  C  |
    +-------+                  +-----+
         |
    SVC  V
        +-----+
        |  4  |------+
        +-----+      |
          |          |
    SVB   V          |
        +-----+      |
        |  2  |------+
        +-----+      |
          |          |
    SVB   V          V     VDB
        +-----+      +-+---------+
        |  1  |----->|3| X.Y.FILE|
        +-----+      +-+---------+
           |
           _
           _
```

Figure 10-14 Name Segment Structure

Logical names which are defined to have parameters make use of a parameter list structure in the logical name segment. The structure is chained to the VCB and is of the following form:

```
 Dec     Hex     *--------------------+--------------------*  --------
  0       0      |        Length      |          0         |
                 +--------------------+--------------------+
  2       2      |  Type for Sublist  | Length of Sublist  |
                 +--------------------+--------------------+  Required
  4       4      |                                         |
                 ~                Parameter                ~
                 ~              Entry Blocks               ~
  2+n     2+n    |                                         |
                 +--------------------+--------------------+  --------
  4+n     4+n    |  Type for Sublist  | Length of Sublist  |
                 +--------------------+--------------------+
  6+n     6+n    |                                         |  Optional
                 ~                Parameter                ~
                 ~              Entry Blocks               ~
  2+n+m  2+n+m   |                                         |
                 *-----------------------------------------*  --------
```

The parameter list contains the following:

BYTE CONTENTS

0 Length of entire structure; the sum of 1 plus two times the number of sublists.

1 Zero.

One or more sets of the following fields:

2 Type for sublist. The type of the parameters in the sublist. Types of parameters are:

0 - System parameters
1 - Spooler parameters
2->7F - Reserved
>80->FF - User IPC parameters

3 Length of sublist. The sum of the lengths of all parameter entry blocks in the sublist, referred to as n.

4-3+n Parameter entry blocks, one for each parameter. Formats of parameter entry blocks are described in subsequent paragraphs.

Three formats are defined for parameter entry blocks, one for each of the parameter sizes. A parameter may be a single-bit binary value, a byte value, or a value of more than one byte. Each parameter format includes a parameter number, and one or two bits that identify the format. The parameter entry block format for a single-bit value is:

```
*----------------+-+-*
| Parameter No.  |1|V|
*----------------+-+-*
```

The parameter entry block contains the following:

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

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

```
*----------------+-+-*
| Parameter No.  |0|0|
+----------------+-+-+
|       Value        |
*--------------------*
```

The parameter entry block contains the following:

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

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

```
*----------------+-+-*
| Parameter No.  |0|1|
+----------------+-+-+
|  Parameter Length  |
+--------------------+
|                    |
~     Parameter      ~
~       Value        ~
|                    |
*--------------------*
```

The parameter entry block contains the following:

BYTE	CONTENTS
0	Parameter number byte: Bits 0-5 - Parameter number, 0 through 63. Parameter numbers need not be assigned or ordered in sequence, but must be unique within the sublist. Bit 6 - 0 Bit 7 - 1
1	Parameter length. The number of bytes required for the parameter value.
2-nn	Parameter value. The numbers or characters of the parameter.

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

10.10.3 Name Manager SVC Preprocessing.

When a Name Manager SVC is issued, control is passed to the SVC decoding routine. This routine buffers the user call block into STA along with the BRO. The decoding routine then passes control to the Name Manager preprocessor, with the BRB on the Name Manager queue. This BRB has the form:

Buffered Request Overhead
User's call block as follows:
SVC code 43, error code
subopcode , flags
other fields depending on the entry type
Space for extra words of Name Manager information

The Name Manager preprocessor works with the SVC buffering routine to buffer call block extensions as needed, depending on the SVC number and subopcode. It also retrieves (from the user's JCA) the ID of the requesting task, the stage number, and the name segment SSB address. It stores these values in the extra words of space for Name Manager information. Before an IOU request reaches the Name Manager, the IOU preprocessor also gathers the stage number and the name segment SSB address and stores them in a three-word block following the standard buffered I/O request block.

10.10.4 Details of Name Manager Modules.

The Name Manager consists of several processors and a short section of code that selects one of these processors depending on the SVC code and the subopcode. The entries are:

NMMAPN	-	Determine Name's Value
NMGNPN	-	Determine Next Pathname of Name's Value
NMSETN	-	Create Logical Name/Assign Synonym
NMAPNV	-	Append Pathname to Present Value of Name
NMDELN	-	Delete Name
NMFLEX	-	Find Lexical Successor
NMPURG	-	Purge Names
NMENS	-	Enter New Stage
NMRTPS	-	Return to Previous Stage
NMGDEL	-	Get Next Descendent Error List Entry
NMGSSZ	-	Get Segment Size
NMSAVE	-	Save Names to a File
NMCTC	-	Notice of Task Bid or Termination
NMIOU	-	Pass SVC to IOU after Resolving Logical Names
NMREST	-	Restore Names from a File

A description of each of the entry processors follows.

NMMAPN - Determine Name's Value

For either synonyms or logical names, this operation returns the value of the name to the requester. If the Name Manager finds the name specified, it returns the value string to the buffer specified by bytes 6 and 7 in the call block. For logical names, any parameters defined for the name are also returned, using the buffer to which bytes 8 and 9 of the call block point. Byte 10 is set to 0. If the logical name is assigned to a set of files that have been logically concatenated, only the first file pathname is returned and byte 11 is set to the value 1. Otherwise, byte 11 is set to 0.

NMGNPN - Determine Next Pathname of Name's Value

This is used to determine the various pathnames in a set of logically concatenated files. Only one pathname is returned by this operation. This pathname corresponds to the pathname number specified in bytes 10 and 11. (The number 0 returns the first pathname, 1 returns the second, and so on.) In addition to returning a pathname, the processor also increments this word (bytes 10 and 11) if the pathname returned is not the last one in the list of concatenated files. If the pathname is the last one, the processor returns 0 to bytes 10 and 11.

NMSETN - Create Logical Name/Assign Synonym

This serves either of two functions, depending on the flag set by the user to tell whether this is a synonym or a logical name operation. If it is a synonym operation, the Assign Synonym operation is performed, using the name as a synonym name and giving that name the specified value. Any previous value for the synonym is replaced with the new value. Stage scope rules (described in paragraphs that follow) are strictly followed in the process of assigning the new value to the name.

If a logical name operation is indicated, a Create Logical Name operation is performed. The specified name is given its value (pathname), and any parameters specified are associated with the logical name. As with synonyms, any previous value for the logical name is replaced, and stage scope rules are strictly followed. If the logical name is for a pathname that is to have its last component filled in by IOU (unique pathname to be autogenerated), space for the pathname to be supplied is reserved at the end of the value field within the table entry. However, the length of the value is recorded as the length specified by the user and does not change until IOU notifies the Name Manager of the full pathname.

NMAPNV - Append Pathname to Present Value of Name

This operation adds to a logical name that represents a set of logically concatenated files. The pathname for the first file and all parameters to be associated with the logical name are specified with the Create Logical Name operation. Additional pathnames may then be appended, one at a time, by using this operation. The additional pathnames are stored separately in the logical name segment to avoid having to move the previous name definition to a location large enough to accommodate the additional pathnames. The additional pathnames are linked to the name definition in the order they are supplied by the user.

NMDELN - Delete Name
Depending on the flag setting, the name is deleted from the synonym table or the logical name table. Any appended pathnames are also deleted. If the name represents a job-local temporary file, a Detach Resource operation is performed on the resolved name. Stage scope rules are strictly followed so that deleting the name does not affect the definition seen by other stages within the same job.

NMFLEX - Find Lexical Successor
This routine searches the synonym or logical name definitions, as specified, for the name (if one exists) that is the immediate alphabetic predecessor or successor of the specified name (pointed to by bytes 4 and 5). Whether the predecessor or successor is found depends on the value of the word at bytes 10 and 11 (0 indicates successor, -1 indicates predecessor). If the desired name is found, its name replaces the specified name (pointed to by bytes 4 and 5) and its value is placed in the buffer pointed to by bytes 6 and 7. If parameters are associated with the name and the parameters buffer pointer (bytes 8 and 9) is nonzero and points to a nonzero-length buffer, the parameters are returned in the buffer. If no name is found to meet the requirements, a null string (zero length) replaces the specified name. If the specified name points to a zero-length string, the alphabetically smallest name and its value is returned if the successor was requested; the alphabetically largest name and its value are returned if the predecessor was requested. If the value pointer originally is zero or points to a zero-length string, any name found is returned as usual, but its corresponding value is omitted.

The name buffer to which the call block points must be large enough to hold the maximum results possible, even though the first byte shows only the length of the string that currently occupies the buffer. This routine cannot check to ensure that the buffer is large enough.

NMPURG - Purge Names
This SVC provides the capability to delete a series of names that are (in some sense) logically related. It searches all names for those that have the first n-1 bytes exactly the same as the first n-1 bytes of the specified name. Each time such a name is found, the nth bytes are compared. If the nth byte of the name is greater than or equal to the nth byte of the specified name, the name found is deleted. Otherwise, the name is left intact. The length of the name deleted may be greater than or equal to the specified name.

NMENS - Enter New Stage

A new stage is created with a stage number equal to the lowest number not currently used in the job, and the requesting task is placed into the new stage. This involves allocating an SDB for the new stage, and placing a new stage number in the TSB of the requestor.

NMRTPS - Return to Previous Stage

This operation returns the task to the stage in which it was running previous to its last Enter New Stage operation. This operation is valid only when the task is returning to a stage from which it issued an Enter New Stage SVC. If the task count of the current stage is greater than one, the task count is decremented and the stage number field is adjusted in the TSB of the requesting task. If the task count is only one, the current stage must be deleted. This requires the following:

* Appending any entries in the current stage's descendant error list onto the descendant error list of the parent stage for the current stage

* Searching the current stage's synonyms for $$CC and, if it is found, building another entry on the parent stage's descendent error list

* Deleting all synonyms and logical names that are defined at the level of the current stage

* Delinking and deallocating the current stage's SDB

The descendent error list is a set of synonyms used for SCI to pass error information from one stage to its parent and to allow a background task to report error information to the foreground SCI. It consists of values for the synonyms $$CC, $$ES, $$FN, $$MN, and $$VT.

Whether or not the stage is to be deleted, the requesting task may also specify a list of synonym names to be passed to the parent stage. The names must be placed back-to-back in a buffer to which bytes 4 and 5 point, with the length of each name (in bytes) immediately preceding the name. The length of the entire list must be specified in the first byte of the buffer.

NMGDEL - Get Next Descendent Error List Entry

This returns the first entry on the descendent error list and then deletes that entry. The entry is returned in the value buffer to which bytes 6 and 7 point and consists of the values that the synonyms $$CC, $$ES, $$MN, $$FN, and $$VT had when the entry was made. Entries are made when a daughter stage that has values defined for these five synonyms terminates. The values are returned back-to-back in the buffer. The first byte of the buffer must be set up by the requester to indicate the length of the buffer. Upon completion of the SVC, the first byte contains the length of the entry returned (0 if none).

NMGSSZ - Get Segment Size
This returns no error if the job contains synonyms; otherwise, an error is returned.

NMSAVE - Save Names to a File
This physically copies all currently accessible names to the file whose pathname is specified in bytes 4 and 5. The file is built with only the names that are linked to the SDB for that stage. The file can be on any disk. The pathname is of the same format used to assign LUNOs.

NMCTC - Notice of Task Creation or Termination
Since the Name Manager must maintain a task count for each stage, task management places an entry on the Name Manager queue whenever a task is bid and whenever a task terminates. Task management does not suspend while this entry is processed. A task inherits its stage number from the creator task. The initial task of a job gets a stage number of zero. The call block format for this call is shown as the name request block (NRB) in the section on data structure pictures. It uses a pseudo-subopcode >0C.

NMIOU - Pass SVC to IOU after Resolving Logical Names
All IOU operations that have an access name as one of their arguments are queued to the Name Manager to be processed and passed to IOU. The Name Manager resolves any logical names to their full pathnames and then passes the BRB to IOU. The Name Manager allocates STA for the full pathname and for any associated parameters and modifies the BRB to point to those structures. When IOU autogenerates a pathname for a logical name, it places the BRB back on the Name Manager queue so that the Name Manager can update the logical name with the full pathname.

NMREST - Restore Names from a File
Names are restored to a name segment from the file pointed to by bytes 4 and 5 of the call block. The name segment ID is returned in bytes 12 and 13 of the call block. Only one stage exists in this segment, stage 0.

10.10.5 Stage Scope Rules.

Stage scope rules are used to ensure that each stage has its own set of synonyms and logical names. These rules also enable one stage to make changes to these names without affecting the values of these same names for other stages. These rules are circumvented only when executing a Return to Previous Stage operation with names specified or when executing an IOU request that involves autocreation of a file whose logical name was previously assigned.

Rule 1

When a previously undefined name is defined, the value definition block is queued to the NDB of the requesting stage only and is not accessible by other stages.

Rule 2

When a name's value (as seen by the requesting stage) is changed, the previous (if any) value definition block in the chain of the requesting stage is discarded and a value definition block for the name is built and queued to the SDB of the requesting stage only and is not accessible by other stages.

SECTION 11

DISK STRUCTURES AND FILE I/O

11.1 OVERVIEW OF FILE MANAGEMENT

File management handles I/O operations directed to disk files. Code in the subsystem runs at either task level or XOP level, depending on the stage of processing and the particular operation involved. As with other I/O operations, handling begins in the request processing modules of the operating system, with some preliminary work handled by the general I/O preprocessor IOPREP; then, action is taken by the file management code itself. The initial work of file management occurs at the XOP level, with the JCA of the calling job mapped into the second segment of map file 0. Processing proceeds until an SVC (or direct I/O) must be issued. At this point, a change is made to the file management task-level code.

11.2 STRUCTURE OF A NEW DISK

Before a disk can be initialized for use by file management, it must be checked for surface defects. This is done by the Initialize Disk Surface (IDS) utility. Any defective tracks found by vendor testing must be entered when the IDS command is executed. The IDS utility does not always find all defective tracks on a disk.

When using the IDS utility, the user has the option of initializing the disk for DNOS use or leaving the disk uninitialized. When left uninitialized for DNOS, the format of the disk after running the IDS utility is as follows:

1. Track 0, sector 0 has all zeros except in the word that shows the state of the disk (SCOSTA). This word has the value 2.

2. Track 0, sector 1 contains a list of bad (defective) tracks. The list consists of pairs of words terminated by a word of zero. The first word of each pair contains the head and cylinder number of the first track of a contiguous set of bad tracks. The head number is in bits 0 through 4 of the word, and the

cylinder number is in bits 5 through 15. The second word of each pair is the number of bad tracks in this contiguous set.

The disk surface is required to be initialized with IDS only once. The disk may then be initialized for use by DNOS with the Initialize New Volume (INV) command as often as needed. The information about defective tracks is preserved when an INV is done, but the information can be extended by specifying additional defective tracks.

The INV utility functions only if the disk has a value of 2 or 3 in SCOSTA of track 0, sector 0. The value of 2 is placed there by the IDS utility, and the value of 3 is placed there by INV. For all other values of SCOSTA, the disk is considered not to have the surface initialized.

11.3 DISK DATA STRUCTURES

File management uses a number of data structures on disk volumes as well as a number of in-memory data structures to process disk I/O requests. The structures on disk include information describing the disk and structures describing each file on the volume. Each of these structures is described in the section on data structure pictures.

Under DNOS, all tracks on disks are initialized in physical records of one sector per record. Note that this record is a disk characteristic and is not the same as the physical record size specified when files are created.

DNOS disks are logically divided into allocatable disk units (ADUs), an integral number of sectors on the disk; the number of sectors per ADU varies according to disk size (see Table 11-1). The number of ADUs is always less than 65,536 (that is, each ADU on the disk can be addressed in a 16-bit word). The number of sectors per ADU is always 1 or a multiple of 3. ADUs are numbered from 0, with the first starting on track 0, sector 0. Table 11-2 shows the capabilities of available disks.

Table 11-1 Format Information for Available Disks

Disk Type	Avail Space (MB)	No. of ADUs	No. of Heads	No. of Cylinders	Sectors /Track	Sectors /ADU	Bytes /Sector
DS10	4.7	16320	2	408	20	1	288
DS25	22.3	25840	5	408	38	3	288
DS31/DS32	2.81	9744	2	203	24	1	288
DS50	44.6	51616	5	815	38	3	288
DS200	169.5	65381	19	815	38	9	288
FD1000	1.15	4004	2	77	26	1	288
CMD 16	13.5	53196	1	806	66	1	256
CMD 80	67.3	44330	5	806	66	6	256
DS80	62.7	40819	5	803	61	6	256
DS300	238.3	62045	19	803	61	15	256
WD800-18	18.5	22311	3	603	37	3	256
WD800-43	43.2	52059	7	603	37	3	256
WD800A-43	42.8	55744	3	871	64	3	256
WD800A-100	99.9	65034	7	871	64	6	256
WD500	4.75	18560	4	145	32	1	256
WD500A	17.0	22208	3	694	32	3	256

Table 11-2 Capabilities of Available Disks

Disk Type	Read With Strobing	Read With Offsets	Variable Inter- leave	Diagnostic Cylinders	Transfer Inhibit	Bad Track Mapping
DS10	NO	NO	NO	NO	YES	NO
DS25	YES	YES	NO	NO	YES	NO
DS31/DS32	NO	NO	NO	NO	NO	NO
DS50	YES	YES	NO	NO	YES	NO
DS200	YES	YES	NO	NO	YES	NO
FD1000	NO	NO	YES	NO	YES	NO
CMD 16	YES	YES	NO	YES	YES	NO
CMD 80	YES	YES	NO	YES	YES	NO
DS80	YES	YES	NO	YES	YES	YES
DS300	YES	YES	NO	YES	YES	YES
WD800-18	YES	NO	NO	YES	YES	NO
WD800-43	YES	NO	NO	YES	YES	NO
WD500	NO	NO	YES	YES	YES	NO
WD500A	NO	NO	NO	YES	YES	NO
WD800A-43	YES	NO	NO	YES	YES	YES
WD800A-100	YES	NO	NO	YES	YES	YES

All disks that have been initialized under DNOS have the following physical layout:

* Track 0, sector 0 -- Contains information about the disk volume, such as the volume name and pointers to the volume directory (VCATALOG). Template SC0 describes the information here.

* Track 0, sector 1 -- Contains a list of bad (physically imperfect) areas on the disk. Each entry is two words: the first word is the address of the first bad ADU; the second word is the address of the last bad ADU. A zero word terminates the list.

* The remainder of track 0 contains disk allocation information in the form of bit maps.

* Track 1, sectors 0 to N-2 -- Optionally reserved for the disk program image loader.

* Track 1, next to last sector -- A copy of track 0, sector 0.

* Track 1, last sector -- A copy of track 0, sector 1.

* Highest numbered (innermost) cylinder -- Diagnostic information. This appears on disk maps as the file .S$DIAG.

* The remaining tracks are available for file allocation.

11.3.1 Volume Information.

The information contained in track 0, sector 0 of all disks initialized under DNOS is called volume information. This block is detailed in the section on data structure pictures as SC0, Sector 0, Track 0.

11.3.2 Allocation Bit Map.

To keep track of which areas on the disk are allocated and which are free, the DNOS disk manager maintains a bit map of allocated ADUs. The bit map is located on 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 (PBMs). Each PBM is located in a separate sector on track 0. The first word of each PBM contains the number of the ADU that begins the largest block of free disk space located in the part of the disk that is mapped by the PBM. Each bit in the remaining 127 words represents an ADU. If a bit is 0, the ADU is free; if a bit is

1, the ADU is allocated or bad. Each PBM contains 127 16-bit words of information and maps 2,032 ADUs.

11.4 FILE STRUCTURES

DNOS supports three file types: relative record files (blocked and unblocked), sequential files, and key indexed files. All file types are based on the unblocked relative record type, with extra system overhead needed to implement sequential and key indexed files. Also, three special types of relative record files are available: program files, directory files, and image files.

In the following discussion of file types and structures, a physical record of a file is the amount of data actually transferred by the operating system during an I/O operation to the file; a logical record of a file is the amount of information the user transfers in one (not multiple) Read or Write SVC call. The ratio of the logical record size to the physical record size is called the blocking factor.

11.4.1 Relative Record Files.

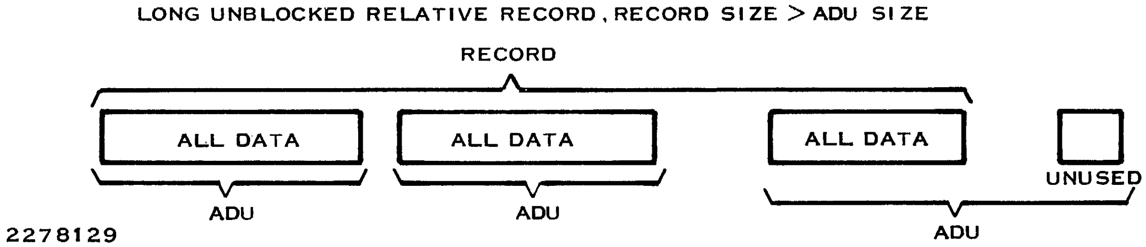
A relative record file is a file in which all logical records are of a fixed length and each record can be randomly accessed by its unique record number. Relative record files may be unblocked (physical record size accommodates only one logical record) or blocked (physical record size accommodates more than one logical record).

11.4.1.1 Unblocked Relative Record Files.

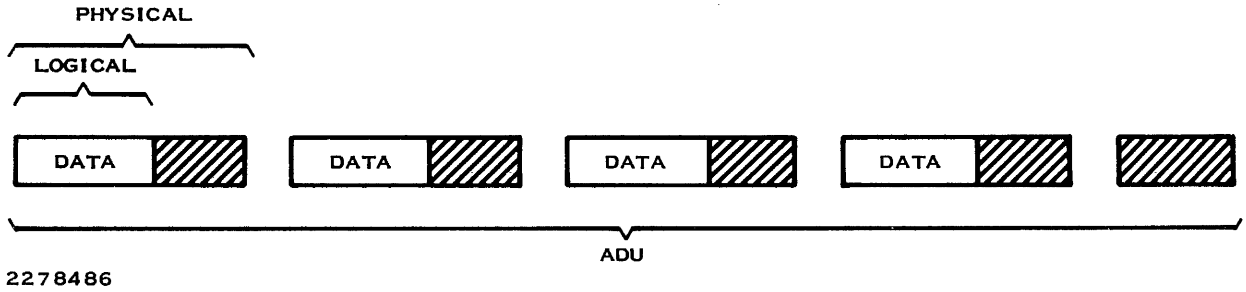
Each logical record of an unblocked relative record file occupies one physical record of the file. A physical record may be any integral multiple of contiguous sectors. File accesses require reading or writing this number of sectors (reads and writes of multiple contiguous sectors can be accomplished via one disk access). 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 by file management to an absolute ADU number and a sector offset within the ADU. The absolute disk address is then passed to the disk DSR to perform the actual data transfer. The disk DSR converts the ADU and relative sector to a physical track and sector disk address to communicate with the disk controller hardware.

The diagrams that follow show examples of long unblocked relative record files and short unblocked relative record files. Assume that the disk in use has 9 sectors per ADU. In the first example, the record might span 3 ADUs, perhaps occupying a total of 25 sectors. Thus, 2 sectors are wasted per physical record. In the second example, each physical record might occupy 2 sectors, wasting 1 sector of each ADU.

Long Unblocked Relative Record File
Record Size > ADU Size



Unblocked Relative Record File
Record Size < ADU Size



Note that each physical record must begin on a sector boundary. Also, a physical record that starts in the middle of an ADU may not span the ADU boundary.

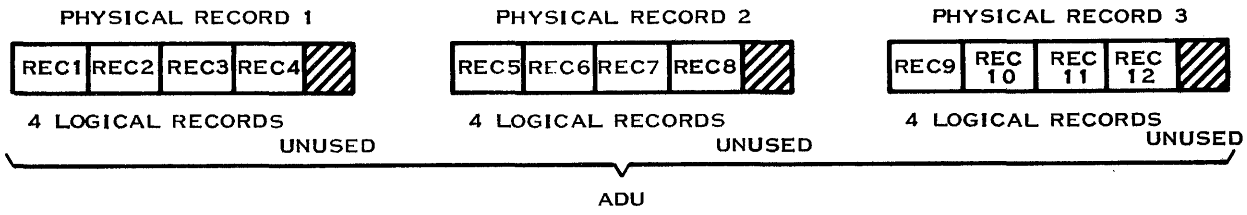
11.4.1.2 Blocked Relative Record Files.

Files of blocked relative records are treated the same as unblocked files except that multiple logical records may be stored in each physical record. Logical records may not span physical records. Records are transferred via intermediate blocking buffers, which are furnished from the general pool of user space by buffer management.

Note that each physical record must begin on a sector boundary and that a physical record that starts within an ADU cannot span the ADU boundary. Also, when physical records are less than an ADU, the number of sectors actually taken up by a physical record is the number of sectors per ADU divided by the number of physical records per ADU.

In the figure which follows, assume that the disk in use has 9 sectors per ADU. If a physical record occupies 3 sectors, three of these physical records would fit into each ADU. Each physical record begins on a sector boundary, so there is no unused space at the end of each ADU, but there may be unused space at the end of each physical record if the logical record size is not an exact multiple of the physical record size. The figure shows each physical record composed of 4 logical records and some unused space.

Blocked Relative Record File



2278485

11.4.2 Sequential Files.

Sequential files are blocked relative record files with variable-length logical records. Logical records may span physical record boundaries regardless of ADU boundaries. When a logical record spans a physical record boundary, it is broken into partial records contained in separate blocks. The first word of each physical record has two flags indicating whether the first logical record is continued from the preceding physical record and whether the last logical record is continued in the following physical record.

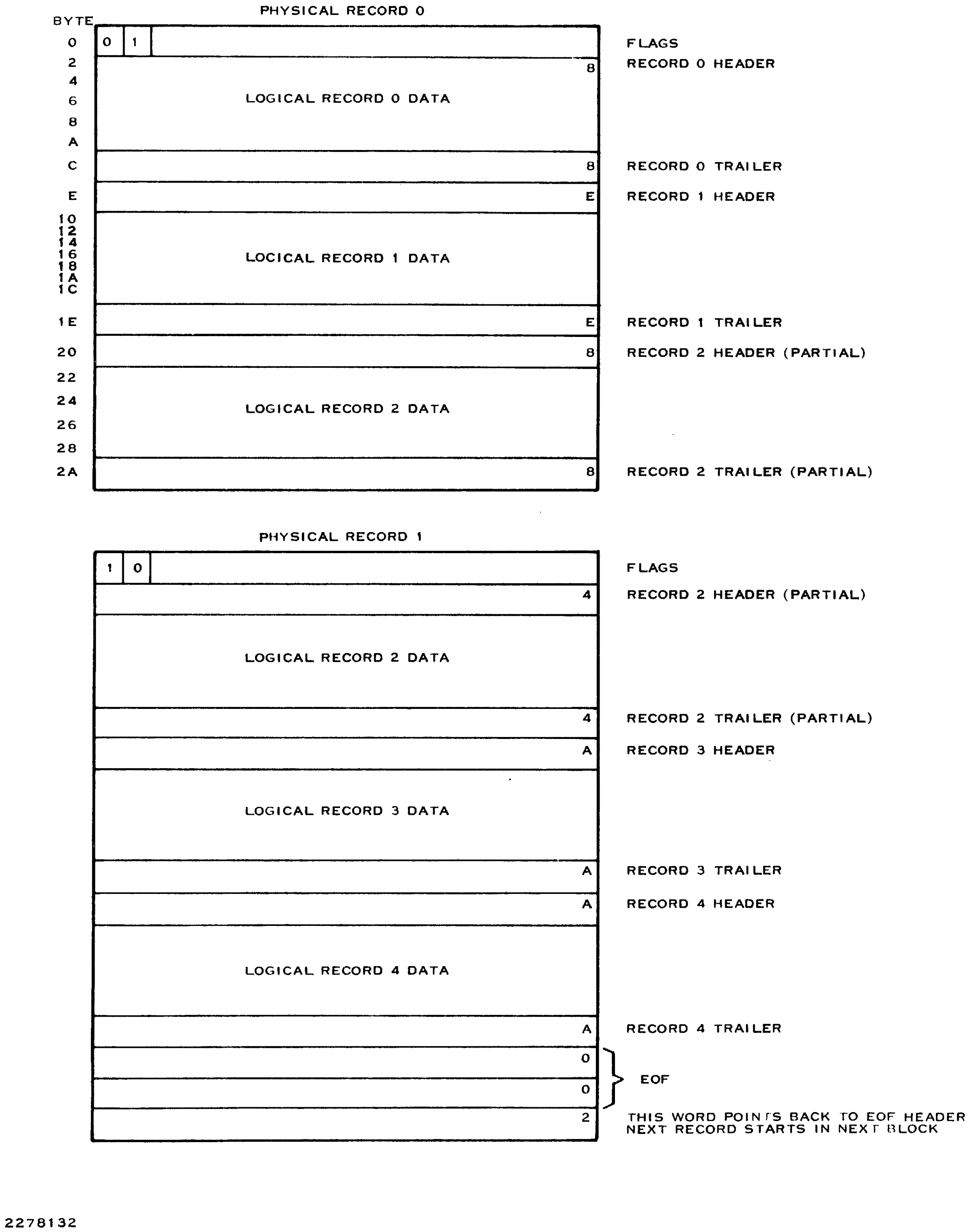


Figure 11-1 Sequential File Format

When set to 1, flag bits have the following meanings:

* Bit 0 - First logical record in this physical record is continued from the preceding record

* Bit 1 - Last logical record in this physical record continues in the next record

Each logical record or partial record is preceded by a header word and followed by a trailer word. The content of the header and trailer is the number of bytes of data between them. An end-of-file is signified by a zero value header and trailer. A zero length record is indicated by a header and trailer containing >FFFF.

A special condition exists when a record or last partial record ends with only one or two words remaining in the physical block. Since there is not room for another partial record (header/data/trailer), the next record begins in the following block. The last word of the current block contains the number in the last trailer plus the number of unused bytes (two or four). Figure 11-1 shows how a sequential file is arranged.

Logical records of a sequential file may be blank-suppressed (defined as blank-suppressed when created). In blank-suppressed files, all full words of blanks are removed. A blank-suppressed logical record includes a header word, a set of data, and a trailer word. The set of data includes one or more repetitions of a byte containing a count of words of blanks, a byte containing a count of words of characters with no words of blanks, and data characters. If a logical record has a length of zero, the physical record shows two words of FFFF. Figure 11-2 shows a blank-suppressed record.

```
Input Record:
  column: 0    0    1    1    2    2    3    3    4    4     50-80
          1    5    0    5    0    5    0    5    0    5
         ------------------------------------------------------
          FIRST          LAST           AGE    (columns 33-80 blank)

Physical Record on File:                    (counts in diagram in hex,
       *-----------------------*             characters in hex ASCII)
       /         0016          /            header record
       +-----------+-----------+
       |    00     |    03     |            (0 words blanks, 3 words data)
       +-----------+-----------+
       |    46     |    49     |              F I
       +-----------+-----------+
       |    52     |    53     |              R S
       +-----------+-----------+
       |    54     |    20     |              T blank
       +-----------+-----------+
       |    04     |    02     |            (4 words blanks, 2 words data)
       +-----------+-----------+
       |    4C     |    41     |              L A
       +-----------+-----------+
       |    53     |    54     |              S T
       +-----------+-----------+
       |    05     |    02     |            (5 words blanks, 2 words data)
       +-----------+-----------+
       |    20     |    41     |            blank A
       +-----------+-----------+
       |    47     |    45     |              G E
       +-----------+-----------+
       |    18     |    00     |            (24 words blanks, 0 words data)
       +-----------+-----------+
       /         0016          /            trailer record
       *-----------------------*
```

Figure 11-2 Blank-Suppressed Record

11.4.3 Key Indexed Files.

Key indexed files have variable-length logical records that can be accessed either randomly, by any of up to 14 keys, or sequentially, in the sort order using any key. On the disk, a key indexed file with n keys is arranged as follows:

* The first 18n+3 (n=number of keys) physical records are the KIF prelog blocks. Before a record in the file is modified, it is written into a prelog block to prevent data loss in case of an error (for example, power failure) during the data transfer or in case the

operation is partially completed when an error occurs. If an error occurs, the logged blocks are written back into the original file record when the file is next opened (in the case of a system crash) or before the operation terminates (in the case of user errors), and the file operation may be retried.

* The next n physical records are the roots of the balanced trees (B-trees) that are used to locate each logical record within the file by key. Every defined key has a corresponding B-tree (up to 14 B-trees); therefore, each key indexed file has n B-tree roots.

* Following the B-tree root nodes are physical records that contain data as well as those that contain other B-tree nodes.

B-trees are made up of a root node, branch nodes, and leaf nodes. A root node is the first node of the tree. Leaf nodes contain pointers to the data records. Branch nodes are all of the nodes between the root and leaf nodes. A root node may be a leaf node, in which case there are no branch nodes.

A DNOS B-tree has multiple branches per node and all leaf nodes are at the same level. DNOS B-trees may not exceed nine levels. Figure 11-3 shows a sample B-tree in which the key values are single letters.

Each node of a B-tree occupies one physical record of a key indexed file, and is called a B-tree block (BTB). Each BTB contains 18 bytes of overhead and several pointer/key value entries. These entries are sorted in increasing order of key value (smallest key value is the first entry).

If the block is not a leaf entry, each pointer field points to a subtree that contains key values less than or equal to the key value associated with the pointer. In fact, the highest key value contained in the subtree is the key value associated with the pointer (as shown in the sample B-tree).

Further information on general B-tree structure is available in The Art of Computer Programming, Volume III by Donald Knuth.

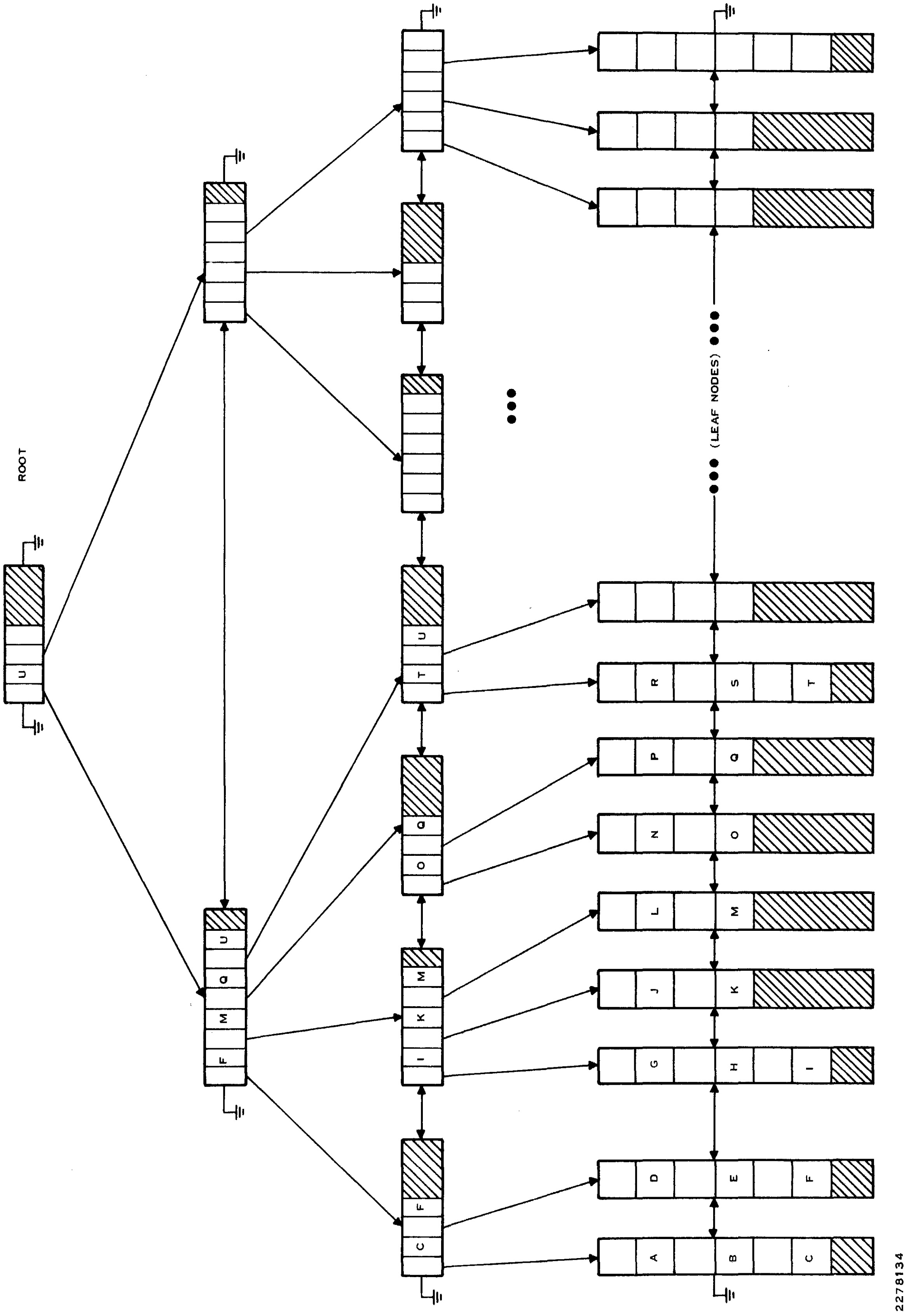


Figure 11-3 Key Indexed File B-Tree

All of the data records (logical records) of a key indexed file are contained in data blocks. A data block is a physical record of the file and contains 14 bytes of overhead and several logical records. The word following the last logical record has a zero value. The structure of a KIF information block (KIB) is shown in the section on data structure pictures.

Whenever a data record is to be-inserted in a data block, it is assigned an ID that is unique within the block. The data record is then inserted after the last logical record in the block.

11.4.4 Program Files.

In addition to the three basic file types, three special uses of the relative record file warrant description: program files, directory files, and image files.

Program files are unblocked relative record files with a logical record size of one sector. The sector size is hardware dependent, with the smallest sector size being 256 bytes. Figure 11-4 shows the format of a program file. The program file directory index entry (PFI) and the program file record zero (PFZ) are shown in detail in the section on data structure pictures.

The sections of information describing the contents of the program file do not always start at the beginning of records or in the same place for all program files. The following equations define the record number and the offset into the record which defines the beginning of the information. In the equations, R designates a record and F designates the offset.

$$R1 = 1$$
$$F1 = 0$$

$$R2 = R1 + \{((\text{MAX \# TASKS} +2)/2) * {>}10) + F1\} / {>}100$$

$$F2 = \text{remainder of } \{((\text{MAX \# TASKS} +2)/2) * {>}10 + F1\} / {>}100$$

$$R3 = R2 + \{(\text{MAX \# TASKS} +1) * {>}10 + F2\} / {>}100$$

$$F3 = \text{remainder of } \{(\text{MAX \# TASKS} +1) * {>}10 + F2\} / {>}100$$

$$R4 = R3 +\{((\text{MAX \# PROCS} +2)/2) * {>}10 + F3\} / {>}100$$

$$F4 = \text{remainder of } \{((\text{MAX \# PROCS} +2)/2) * {>}10 + F3\} / {>}100$$

$$R5 = R4 +\{(\text{MAX \# PROCS} +1) * {>}10 + F4\} / {>}100$$

$$F5 = \text{remainder of } \{(\text{MAX \# PROCS} +1) * {>}10 + F4\} / {>}100$$

$$R6 = R5 +\{((\text{MAX \# OVLYS} +2)/2) * {>}10 + F5\} / {>}100$$

F6 = remainder of {((MAX # OVLYS +2)/2) * >10 + F5} / >100

F7 = R6 +{(MAX # OVLYS +1) * >10 + F6} / >100

F7 = remainder of {(MAX # OVLYS +1) * >10 + F6} / >100

R8 = R7 + {((MAX # HOLES * 4) +2) + F7} />100

F8 = remainder of {((MAX # HOLES * 4) +2) + F7} / >100

If F8 is not equal to zero, then R8 = R8 + 1

R1,F1: Record number and offset for names of tasks.
R2,F2: Record number and offset for task directory entries.
R3,F3: Record number and offset for names of procedures.
R4,F4: Record number and offset for procedures directory entries.
R5,F5: Record number and offset for names of overlays.
R6,F6: Record number and offset for overlay directory entries.
R7,F7: Record number and offset for unused space directory.
R8: Record number of first image record.

The first record (record number 0) of a program file contains six bit maps. These bit maps, in order of occurrence within record 0, are for memory-resident tasks, memory-resident procedures or segments, all tasks, all procedures, all nonreplicatable tasks, and all overlays.

```
        *------------------------------------------------------*
   0    |                  OVERHEAD RECORD                     |
        +------------------------------------------------------+
   1    |               NAME ENTRIES FOR TASKS                 |
        +------------------------------------------------------+
R2:02   |            DESCRIPTION ENTRIES FOR TASKS             |
        +------------------------------------------------------+
R3:03   |        NAME ENTRIES FOR PROCEDURES/SEGMENTS          |
        +------------------------------------------------------+
R4:04   |     DESCRIPTION ENTRIES FOR PROCEDURES/SEGMENTS      |
        +------------------------------------------------------+
R5:05   |              NAME ENTRIES FOR OVERLAYS               |
        +------------------------------------------------------+
R6:06   |           DESCRIPTION ENTRIES FOR OVERLAYS           |
        +------------------------------------------------------+
R7:07   |                AVAILABLE SPACE LIST                  |
        +------------------------------------------------------+
R8      |    IMAGE FORMATS FOR TASKS, PROCEDURES AND OVERLAYS  |
        *------------------------------------------------------*
```

Figure 11-4 Program File Format

When record 0 is initialized, all bits in the bit map are 0 except the first bit in the tasks, procedures/segments, overlays, and nonreplicatable tasks bit maps (the bit maps occupying bytes >54 through >D3). The first bit of these is a 1, restricting user tasks from allocating ID 0.

Each bit map has 16 words, with 16 bits per word; therefore, each bit map can represent 256 IDs. A bit set to 1 indicates that the ID corresponding to the bit position (0 through 255) is assigned to a task, procedure, or overlay segment installed in the file. The format of record 0 of the program file is shown in the section on data structure pictures as PFZ, Program File - Record Zero.

When a program file is created, the maximum number of tasks, procedures/segments, and overlays to be set into bytes >D4, >D8, and >DC of record 0 are defined by the creator of the program file. The maximum number of holes, which equals the sum of these three values, is used to calculate the number of bytes required in the overhead records for the available space list. This list is headed by a word containing the number of entries in the list. The rest of the list consists of two-word entries that describe the unallocated spaces (holes) in the image portion of the program file. Each entry contains the starting record number and the number of available records in each hole. A hole appears when an image is deleted. The hole is recorded to be used again if a new image that is the same size or smaller than the deleted one is installed in the file. Adjacent images, when deleted, create only one hole. Figure 11-5 shows the format of the available space list.

```
*--------------------------------------------------------*
|                  NUMBER OF ENTRIES                     |
+--------------------------------------------------------+--+
|                    RECORD NUMBER                       |  |
+--------------------------------------------------------+ ENTRY 1
|                  RECORDS AVAILABLE                     |  |
+--------------------------------------------------------+--+
|                          .                             |
                         .                             /
|                                                        |
+--------------------------------------------------------+--+
|                    RECORD NUMBER                       |  |
+--------------------------------------------------------+ ENTRY n
|                  RECORDS AVAILABLE                     |  |
*--------------------------------------------------------*--+
```

Figure 11-5 Program File Available Space List

The available space list uses the entire record, not 256 bytes of it as the other overhead records do. Therefore, if the list spans records, an entry is split across two records. (The first word of the entry is the last word of one record, and the second word of the entry is the first word of the next record.) The available space list is initialized at the same time record 0 is initialized. Its values are as follows:

```
*---------------------------*
|             1             | ONE HOLE
+---------------------------+
|            R8             | BEGINS AT RECORD 8
+---------------------------+
|         >FFFF-R8          | IS >FFFF - R8 RECORDS LONG
*---------------------------*
```

R8 is the record number of the first record following the available space list.

The maximum number of records permitted in a program is >FFFF. Thus, the maximum number of image records permitted in a program file is >FFFF minus the number of overhead records.

The actual image of a task, procedure, or overlay must start on a record boundary in the program file. If the segment has a relocation bit map, the map begins at the first word following the program segment image. However, any part of a program file can be split across secondary allocations. The relocation bit map begins at the first word following the program segment image. The length of the relocation bit map is the length of the program segment image, in bytes, divided by eight and rounded to a word boundary

The task, procedure/segment, and overlay name entries in the program file contain the names of all tasks, procedures/segments, and overlays installed in the program file. A name entry is eight bytes long, blank-filled to the right. The name entry is placed in the name block position that corresponds to the ID assigned to that segment. For example, if task GENTX is assigned ID 1, the name GENTX is entered in bytes 8 through 15 (second position) of the name entries block for tasks.

The task, procedure/segment, and overlay description entries in the program file contain information about all segments installed in the program file as well as pointers to the segment images. Each description is 16 bytes long. The figures that follow show the formats of the program file description entries, with field descriptions following each format. Figure 11-6 shows the format of the task description; Figure 11-7 shows the format of a procedure/segment description; and Figure 11-8 shows the format of an overlay description.

```
Hex.
Byte
-----  *---------------------------------------------------*
 >00   |              LENGTH OF TASK SEGMENT                |
       +---------------------------------------------------+
 >02   |                      FLAGS                         |
       +---------------------------------------------------+
 >04   |                  RECORD NUMBER                     |
       +---------------------------------------------------+
 >06   |                  DATE INSTALLED                    |
       +---------------------------------------------------+
 >08   |                   LOAD ADDRESS                     |
       +-------------------------+-------------------------+
 >0A   |      OVERLAY LINK       | PRIORITY OF THE TASK    |
       +-------------------------+-------------------------+
 >0C   |    PROCEDURE 1 ID       |    PROCEDURE 2 ID       |
       +-------------------------+-------------------------+
 >0E   |                   TASK LENGTH                      |
       *---------------------------------------------------*
 >10   maximum size
```

Figure 11-6 Task Description Entry

Hex.
Byte Description of Selected Fields

>00 Length of task segment in bytes. Length of task root plus length of the task's longest overlay path.

>02 Flags, as follows:

Bit	Meaning When Set
0	Privileged
1	System
2	Memory resident
3	Delete protected
4	Replicatable
5	Procedure 1 is on the system program file
6	Procedure 2 is on the system program file
7	Directory entry in use
8	Overflow
9	Writable control store (WCS)
10	Execute protected
11	Software privileged
12	Updatable
13	Reusable
14	Copyable
15	Security bypass

>04 Record number. Logical record number of the start of the task image in the program file.

>06 Date installed, in the following format:

Bit	Meaning
0-6	Year (displacement)
7-15	Julian date

>08 Load address. Relative starting address within a mapped task segment. Must be on a beet boundary.

>0A Overlay link. The ID of the most recently installed overlay associated with the task. Each overlay entry is in turn linked to the next entry so that tasks can be associated with their overlays when status or delete commands are executed. A value of 0 is used to terminate the list.

>0E Task length. Last defined task code. If a BSS is the last instruction in the task, its length is not included in the value.

```
Hex.
Byte
----- *----------------------------------------------------*
 >00  |          LENGTH OF PROCEDURE/SEGMENT (BYTES)         |
      +----------------------------------------------------+
 >02  |                       FLAGS                          |
      +----------------------------------------------------+
 >04  |                   RECORD NUMBER                      |
      +----------------------------------------------------+
 >06  |                   DATE INSTALLED                     |
      +----------------------------------------------------+
 >08  |                    LOAD ADDRESS                      |
      +----------------------------------------------------+
 >0A  |                                                      |
      ~                    UNUSED (=0)                       ~
      |                                                      |
      *----------------------------------------------------*
 >10  * maximum size
```

Figure 11-7 Procedure/Segment Description Entry

Hex.
Byte Description of Selected Fields

>02 Flags, as follows:

Bit	Meaning When Set
0	Unused (set to zero)
1	System (segment only)
2	Memory resident
3	Delete protected
4	Replicatable (segment only)
5	Share protected
6	Unused (set to zero)
7	Directory entry in use
8	Unused (set to zero)
9	Writable control store(WCS)
10	Execute protected
11	Write protected
12	Updatable (segment only)
13	Reusable (segment only)
14	Copyable (segment only)
15	Unused (set to zero)

>04 Record number. Logical record number of the start

of the procedure image in the program file.

>06 Date installed, in the following format:

Bit	Meaning
0-6	Year (displacement)
7-15	Julian date

>08 Load address. Relative starting address within a mapped procedure segment. Must be on a beet boundary.

```
Hex.
Byte
----- *-----------------------------------------------------*
 >00  |            LENGTH OF OVERLAY SEGMENT (BYTES)          |
      +-----------------------------------------------------+
 >02  |                         FLAGS                         |
      +-----------------------------------------------------+
 >04  |                     RECORD NUMBER                     |
      +-----------------------------------------------------+
 >06  |                     DATE INSTALLED                    |
      +-----------------------------------------------------+
 >08  |                      LOAD ADDRESS                     |
      +--------------------------+--------------------------+
 >0A  |  LINK TO NEXT OVERLAY     |  ID OF ASSOCIATED TASK    |
      +--------------------------+--------------------------+
 >0C  |                      UNUSED (=0)                      |
      |                                                       |
      *-----------------------------------------------------*
 >10  * maximum size
```

Figure 11-8 Overlay Description Entry

Hex. Byte ---- Description of Selected Fields

>02 Flags, as follows:

Bit	Meaning When Set
0	Relocation bit map is present
1-2	Unused (set to zero)
3	Delete protected
4-6	Unused (set to zero)
7	Directory entry in use
8-15	Unused (set to zero)

>04 Record number. Logical record number of the starting

address of the overlay image in the program file.

>06 Date installed, in the following format:

Bit	Meaning
0-6	Year (displacement)
7-15	Julian date

>08 Load address. Relative starting address within a mapped overlay segment. Must be on a beet boundary.

11.4.5 Directory Files.

Directory files are unblocked relative record files and have a record length of >86 or >100 characters. Record 0 of the directory file contains an overhead record. The remaining records in the file may contain one of the following types of data blocks:

* File Descriptor Record (FDR) -- Every file cataloged in the directory is represented by an FDR, which describes the file and its location on the disk.

* Alias Descriptor Record (ADR) -- Every alias of a file cataloged in the directory is represented by an ADR, which gives the location of the file and points to the FDR of the actual file.

* Channel Descriptor Record (CDR) -- Every channel that has an owner task in a program file in the directory is represented by a CDR, which describes the channel characteristics and identifies its owner task.

* Key Descriptor Record (KDR) -- Each key indexed file cataloged in the directory is represented by an FDR, which in turn points to another record, the KDR. The KDR describes all of the keys (1 through 14) that are defined for the file. Note that the use of the KDR implies that each key indexed file cataloged in a directory uses two directory entries.

Figure 11-9 shows the general structure of a directory file. Entries are made by hashing the name of the file being entered. The hash algorithm results in a record number from 1 through n, where n is the last record in the directory file. Figure 11-10 shows the hash algorithm. If the directory file record is unused, an FDR for the file being inserted is placed in that record. If the record is already used, a free record is found by a linear search from the hashed record.

```
Record No.
         *----------------------------*
   0     |      OVERHEAD RECORD       |
         +----------------------------+
   1     |                            | \
         +----------------------------+  |
   2     |                            |  |
         +----------------------------+   > DIRECTORY ENTRIES
         ~                            ~  |
         +----------------------------+  |
   n     |                            | /
         *----------------------------*
```

Figure 11-9 Directory File Structure

```
PROCEDURE HASH (N : number of records in directory minus 1,
                NAME : name of the file being entered)
  BEGIN
  KEY := 1;
  I   := 1;
  C   := NAME[I];
  WHILE C <> ' ' AND I < 9 DO
    BEGIN
    KEY := ((KEY * C) MOD N) + 1;
    I := I + 1;
    C := NAME[I];
    END
  END
```

Figure 11-10 Computing a Hash Key

If the file being inserted is a key indexed file, another directory record must be found to contain the KDR. This record is found by searching linearly from the FDR for the file. The KDR is inserted into the first available directory record following the FDR.

The different types of directory records are described in the following paragraphs.

The directory overhead record (DOR), which is record 0 of all directories, contains:

* The maximum number of records (entries) in the directory

* The number of currently defined files

- * The number of available records (entries)
- * The file name of the directory
- * The level number of the directory in the disk hierarchy (VCATALOG) is level 0)
- * The file name of the parent directory
- * The default physical record length

Each file cataloged under the directory is represented by an FDR.

Files can be given other names, each name being a separate alias. Each alias is hashed to find an entry in the directory just like a file name, and an ADR is inserted in that entry. The ADR points to the actual file. It also points to the next alias for the file.

Figure 11-11 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 key indexed file (.JF.DIR.KIF). The directory also contains an alias for the key indexed file. The directory was created to have 11 entries in addition to record 0 which is the DOR.

11.4.6 Image Files.

Image files are contiguous nonexpandable, unblocked relative record files that contain memory images of programs. They are not organized in any format; that is, each sector of the image file, starting with the first sector, is completely filled with data. There are no overhead records or words. Image files are designed so that a program image can be read into memory in a single disk access.

11.5 ALLOCATION OF SPACE FOR EXPANDABLE FILES

When a file must be expanded, the amount allocated for the expansion depends on how much space is needed and where the space is available on the disk. If there is a space available contiguous with the last allocation of the file, that space is allocated for the expansion. In this case, the amount of space allocated may be less than that asked for by the file definition. If space is not available contiguous with the current file allocation, one of two secondary allocations is made. If a contiguous block is available with the size requested, that block is allocated. Otherwise, the largest available contiguous block of space is allocated as the secondary allocation.

The amount of space asked for initially is the larger of

SAS * (2 ** #SA)

and

TIMTBL(#EXT) converted to ADUs

where:

SAS is the defined secondary allocation size in ADUs
#SA is the number of noncontiguous secondary allocations (ranging from 0 through 16)
#EXT is the number of file extensions, ranging from 0 through 15, initialized to #SA when a LUNO is assigned
TIMTBL(0) is 1 physical record
TIMTBL(1) is 2 physical records
TIMTBL(2) is 4 physical records
TIMTBL(3) is 8 physical records
TIMTBL(4) is 12 physical records
TIMTBL(5) is 16 physical records
TIMTBL(6) is 20 physical records
TIMTBL(7) through TIMTBL(15) are 24 physical records

Each type of file can have secondary allocations. An image in a program file can also have a secondary allocation.

```
FILE ACCESS NAME:  .JB.DIR
RECORD:  000000
0000  000B 0004 0005 0000 4449 5220 2020 2020     .. .. .. .. DI R
0010  0002 4A42 2020 2020 2020 0360 0000 0000     .. JB          . .. ..
  SAME
00FE  0000                                        ..
RECORD:  000001
0000  0002 0001 5345 5120 2020 2020 0000 0000     .. .. SE Q         .. ..
0010  0A00 0360 0050 0001 2259 0001 0000 0003     .. .  .P .. ". .. .. ..
0020  0000 0000 0000 0000 0000 0000 0000 0000     .. .. .. .. .. .. .. ..
0030  0000 0000 0000 07BC 0203 095B 07BC 0203     .. .. .. .< .. .[ .< .
0040  095B 0101 0000 0000 0000 0000 0000 0000     .[ .. .. .. .. .. .. .
  SAME
0090  4A4F 5943 4520 2020 0000 0000 0000 0000     JO YC E     .. .. .. .
  SAME
00FE  0000                                        ..
RECORD:  000002
0000  0000 0001 494D 4147 2020 2020 0000 0000     .. .. IM AG        .. .
0010  C420 0120 0120 0022 3952 0001 0000 0000     D     .  ." 9R .. .. ..
  SAME
0036  07BC 0203 09D6 07BC 0203 09D6 0103 0000     .< .. .V .  .. .V .. ..
  SAME
0090  4A4F 5943 4520 2020 0000 0000 0000 0000     JO YC E     .. .. .. ..
  SAME
00FE  0000                                        ..
RECORD:  000003
0000  0001 0003 414C 4941 5320 2020 0000 0000     .. .. AL IA S      .. ..
0010  0A10 0000 0000 0000 0000 0000 0000 0000     .. .. .. .. .. .. .. ..
0020  0001 0000 0000 0000 0000 0000 0000 0000     .. .. .. .. .. .. .. ..
  SAME
00FE  0000                                        ..
RECORD:  000004
0000  0000 0000 0000 0000 0000 0000 0000 0000     .. .. .. .. .. .. .. ..
  SAME
00FE  0000                                        ..
RECORD:  000005
0000  0000 0000 0000 0000 0000 0000 0000 0000     .. .. .. .. .. .. .. ..
  SAME
00FE  0000                                        ..
RECORD:  000006
0000  0000 0000 0000 0000 0000 0000 0000 0000     .. .. .. .. .. .. .. ..
  SAME
00FE  0000                                        ..
RECORD:  000007
0000  0001 0007 5052 4F47 2020 2020 0000 0000     .. .. PR OG        .. ..
0010  8C20 0120 0120 001D 3B2A 0001 0000 0000     .  .  .  .. ;* .. .. ..
0020  0000 004A 0000 004A 0000 0000 0000 0000     .. .J .. .J .. .. .. ..
0030  0000 0000 0000 07BC 0203 0BFC 07BC 0203     .. .. .. .< .. .. .< ..
0040  0BF6 0103 0000 0000 0000 0000 0000 0000     .. .. .. .. .. .. .. ..
  SAME
0090  4A4F 5943 4520 2020 0000 0000 0000 0000     JO YC E     .. .. .. ..
  SAME
00FE  0000                                        ..
RECORD:  000008
0000  0000 0000 0000 0000 0000 0000 0000 0000     .. .. .. .. .. .. .. ..
  SAME
00FE  0000                                        ..
RECORD:  000009
0000  0001 0009 4B45 5946 494C 4520 0000 0000     .. .. KE YF IL E  .. ..
0010  1E08 00A0 0050 00B0 72B9 0022 0000 0000     .. .. .P .0 r9 ." .. ..
0020  0000 0000 0000 0016 0001 0000 0016 0015     .. .. .. .. .. .. .. ..
0030  0016 01F7 000A 07BC 0203 0C23 07BC 0203     .. .. .. .< .. .# .< ..
0040  0C23 0103 000A 0000 0000 0000 0000 0000     .# .. .. .. .. .. .. ..
  SAME
0090  4A4F 5943 4520 2020 0000 0000 0000 0000     JO YC E     .. .. .. ..
  SAME
00FE  0000                                        ..
RECORD:  00000A
0000  0000 FFFD 01F4 0001 080A 0000 0000 0000     .. .. .. .. .. .. .. ..
  SAME
00FE  0000                                        ..
```

Figure 11-11 Dump of Directory File

11.6 IN-MEMORY DATA STRUCTURES

The primary data structure used by file management during its execution is a record of the state of the request being processed, the File Manager work area (FWA). With each FM request, extra storage is allocated for the FWA when the IRB is processed by FMPREP. The extra storage is required for the following information:

* A workspace for execution

* Information about the request including block numbers, offsets, several fields of the FCB needed when the FCB is not available, a description of the user buffer, a description of the blocking buffer, and vectors used to terminate processing at XOP level and reactivate at task level

* A stack in which called routines save registers

The part of the buffered request allocated for working storage is pointed to by register 15 (R15), except in FMPREP and FMTASK, and is addressed with the template FWA. The FWA is detailed in the section on data structure pictures, as are the other in-memory structures.

Other in-memory structures used by file management include the following:

* File Control Block (FCB) - In-memory structure representing the last component of a file pathname, used to access the file for direct disk I/O in conjunction with the file directory block

* File Directory Block (FDB) - Single node of the in-memory directory tree structure located in the file management table area. Provides tree linkage and information needed to perform direct disk I/O to the file

* File Descriptor Packet (FDP) - A two-word in-memory structure that is used to access an FCB

* Logical Device Table (LDT) - A description of the file resource, including usage flags, ownership information, parameters, and a pointer to the relevant FDP

* Resource Privilege Block (RPB) - A structure used to control access privileges for resources

Figure 11-12 shows the location and relationships between the various file structures maintained in memory in DNOS.

```
        File Management Table Area
     +------------------------------+
     |  FDB                         |
     | +--------+                   |
     | |VCATALOG|<----------------| |
     | |        |                 | |
     | +--------+  FDB      FDB   | |
     |        |   +---+    +---+  | |
     |   |    --->|for|--->|for --| |
     |   ---------| A |<---| B |    |
     |  FDB       +---+    +---+    |
     | +---+        |        |      |
   |-->|for|<-------|        |      |
   | | | C |                 |      |
   | | +---+                 |      |
   | +-----------------------|------+
   | |                       |      |
   | |                       |      |
   | |                       |      |
   | +----------------------+|      |
   | | |    FCB             ||      |
   +-|---+-------+  +------+-+      |
     |   |  for  |  |  for ||       |
     |   | .A.C  |  |  .B  ||       |
     |   +-------+  +------+|       |
     +------------------|---+-------+
+------------------------
|    |    Segment Manager Table Area
|    | +-------------------------------+
|    | |  SGB      SSB            SSB  |
|    | | +---+    +---+          +---+ |
|    --->|   |--->|   |---...--->|   | |
|      | +---+    +---+          +---+ |
|      |                               |
|      |  SGB      SSB            SSB  |
|      | +---+    +---+          +---+ |
-------->|   |--->|   |---...--->|   | |
       | +---+    +---+          +---+ |
      +--------------------------------+
```

Figure 11-12 In-Memory File Representation

File management routines fall into the following categories:

* XOP-level preprocessing

* Task-level preprocessing

* Address space management, including transfers between XOP-level and task-level code and the management of overlays

- Buffer management

- Routines to perform file I/O operations (IODRCT)

- Low-level routines that support I/O SVC sub-opcode processing

File management code resides in four different areas. Some code is found in the map 0 segment SVCSHD with other SVC processors, some is in the variable part of the operating system root, some is in the file management task segment, and some is found in overlays on disk.

Table 11-3 shows the major modules found in the file management source directory and where they fit into the six major categories of routines. In addition to those listed, modules of stub routines are included for those systems that do not support particular file management options. The names of these modules include an S after the functional module name (for example, FMBKOFS is the stub for FMBKOF). Stubs are found in modules FMBKOFS, FMCKEXS, FMFBSQS, FMOPSXS, FMRDBFS, FMRDUBS, FMRWSQS, FMSQORS, FMWRBFS, FMWRUBS, and KMBEGS. The last stub is for systems that do not support KIF.

Table 11-3 File Management Modules

XOP-Level Preprocessing		
	FMACTV	Driver for opcode processors
	FMPREP	XOP-level preprocessing - base routine
Task-Level Preprocessing		
	FMTASK	Task-level driver
Address and Overlay Management		
	FMOVL0	Overlay header tables, one for each overlay
	FMOVL1	Overlay header tables, one for each overlay
	FMOVL2	Overlay header tables, one for each overlay
	FMOVL3	Overlay header tables, one for each overlay
	FMOVLYC	Overlay code location tables - memory-resident systems
	FMOVLYD	Overlay code location tables - disk-resident systems
	FMPMGR	Set of overlay pool management routines
	FMTRAN	Transfer at XOP level between SVCSHD and FMTASK
Buffer Management		
	FMBIO	Set of routines to buffer I/O and map blocked files
	FMCPB	Copy buffer - blocked file
	FMCPBI	Interface routine for FMCPB

Table 11-3 File Management Modules (Continued)

I/O SVC Sub-opcode Processors	
FMCLEF	Close with EOF processor
FMCLOS	Close operation processor
FMFBSP	Forward Space/Backspace operation processor
FMFBSQ	Forward Space/Backspace processor for sequential files
FMOPEN	Open operation processor
FMOPRW	Open Rewind operation processor
FMOPSX	Open Extend sequential file operation processor
FMOPUB	Unblocked Open operation processor
FMOPXT	Open Extend operation processor
FMRDF	Read operation processor
FMRDUF	Read Unblocked operation processor
FMRWF	Rewrite operation processor
FMSQOR	Open Rewind for sequential files
FMLKUL	Unlock operation processor
FMWEOF	Write with EOF operation processor
FMWTF	Write operation processor
FMWTUF	Write Unblocked operation processor
Lower Level Support Routines	
FMBKAD	File extension
FMBKOF	Block number computation
FMBLAJ	Blank adjustment
FMBSRT	Blank suppression routines
FMCKEX	Check file extension
FMEXFL	File extension
FMMREC	Compute ADU and sector from block number
FMFIO	Unblocked relative record I/O interface to FMIO
FMIO	Read/write file block
FMLKNF	Lock operation processor; set of routines to check for locks; add them, and remove them
FMLSET	Creates LUNO for FMBIO on first buffer I/O of each FM task
FMRDBR	Read blocked relative record files
FMRDSQ	Read sequential file
FMRDST	Read file status
FMRDUB	Unblocked read of blocked files
FMRWBC	Rewrite blank compressed counter
FMRWSQ	Rewrite sequential file
FMUPFD	Update FDR
FMUTLY	Set of routines to check EOM; set parameters, and find structures
FMWTBR	Write blocked relative record file
FMWTCK	Check write privileges on a file
FMWTSQ	Write sequential files
FMWTUB	Unblocked write of a blocked file

To avoid the extra overhead required for preprocessing SVCs and I/O calls, a special interface to I/O, FMIO, is used to perform file reads and writes. This same interface is used by the task loader to perform reads and writes to the swap file, and by IOU and the File Manager to update FDR records. The calling program obtains a block of STA sufficient to buffer the disk I/O call block and initializes it as follows:

```
BROOFL = SMT address for segment containing buffer
BROBBA = SSB address for segment containing buffer
BROLDT = PDT address for device desired
BRORCB = 0   (supplied by IODRCT)
BROTSB = EXTSB
BROJSB = EXJSB
BROBRO = 0
IRBSOC = 0   SVC opcode/error code
IRBOC  = 09/0B subopcode/0 (no LUNO)
IRBSFL = 0   system/user flags = 0
IRBDBA = offset into buffer segment
IRBICC = character count
IRBOCC = character count
IRBRN1 = ADU of disk device to be read/written
IRBRN2 = sector in ADU to be read/written
```

The call block built is passed to the direct I/O routine; IODRCT. It then proceeds like an I/O SVC, but the overhead of SVC processing has been avoided.

11.6.1 XOP-Level Preprocessing.

FMPREP initiates XOP-level preprocessing for the File Manager, and resides in SVCSHD. It takes the I/O call block buffered by IOPREP and creates the FWA needed by the File Manager to execute at XOP-level.

The routines in FMTRAN (File Manager task call, FMTCAL, and File Manager task return, FMTRTN) change map file 0 to map in the File Manager task segment so that it can execute at XOP level. When XOP-level work completes, control is transferred back to FMPREP; consequently, the map file must be changed to its previous state. Some requests can complete execution at XOP level without changing to task-level code. The following conditions must be met for execution of a file management request to complete at XOP level; such execution is referred to as fast transfer.

- If the operation is active (such as write, rewrite), no other operations are outstanding for the FCB; if the operation is passive (such as read), no active operation is outstanding for the same FCB.

- No requests are outstanding for the LUNO (that is, the initiate I/O count must be zero).

* The overlay area is available when the opcode requires a system overlay.

* If the operation is Read or Write, the file is a blocked file.

* The LDT does not have the unblocked I/O flag set.

* If the request is a Write, the LDT does not have the forced. write bit set.

* The required blocks are in memory.

A file management request begins execution at XOP level. If a transfer to task-level code is required, the routine FMTSET from module FMUTLY is called. Such conditions can be detected at several points, both in FMACTV and in the operation processor. The following processing occurs at the level indicated:

1. (XOP) IOPREP sets the busy flag in the IRB prior to calling FMPREP. FMPREP passes control to FMACTV, which can call FMTSET to enter task mode. Other routines may also call FMTSET.

2. (XOP) FMPREP sets up the vector in FWAXWP (in the FMT) such that a BLWP instruction transfers control to FMTRTN (in FMUTLY).

3. (XOP) FMTSET checks to see if execution is already in task mode. If so, it exits. If not, it executes a BLWP instruction through FWAXWP (R15). The only purpose of the error is to enable the caller to determine the previous execution mode.

4. (XOP) The BLWP executed by FMTSET takes control to FMTRTN, which changes the map 0 file back to SVCSHD and branches into FMPREP at label FMFXRT, which checks the busy flag in the IRB. If the flag is clear, the request is complete and the call block is directly unbuffered. Initiate event flags are checked; if one is set, the event is marked in the TSB as completed. Upon completion of the unbuffering, FMFXRT returns via a branch to NFTRTN.

5. (XOP) If the request is not complete, the value in R14 from the BLWP is stored in FWAPC (in the FWA). The busy flag is still set, causing FMPREP to queue the operation for task-level file management and exit through IORTN. The busy flag causes IORTN to suspend the calling task (if this is not initiate I/O) and exit to the scheduler.

6. (TASK) Eventually, either the scheduler chooses a file management task for execution (starting at FMTASK), or a

file management task already in execution dequeues the operation. If a buffer is required, the segment of the user program containing the buffer is also loaded. The JCA is mapped in.

7. (TASK) FMTASK puts the contents of FWAPC into its R14, computes the WP address in R13, and puts its own status in R15. The FWAXWP and FWAXPC vector is set up to point to FMP210 in FMTASK. An RTWP is executed, thus resuming the activity in FMTSET immediately following the BLWP that transferred control to FMTRTN. FMTSET sets no error and exits.

11.6.2 Task-Level Processing.

FMTASK is the driver routine that processes the start-up procedure and handles the processing between requests at task level. FMTASK is activated by the operating system queue server mechanism. When a file management task begins, the STA is mapped into the first segment, the user job JCA into the second segment, and the file management code into the third segment. FMTASK uses a workspace and stack that is allocated the first time the task is bid. That workspace is associated with the task, not with a BRB, and is used for the dequeuing and queuing calls and for acquiring the execution environment needed by the activity. FMTASK transfers execution to the point of suspension in XOP-level processing by activating the workspace in the BRB via an RTWP instruction.

File management overlays are read into and executed in a pool of overlay areas allocated with the file management task segment. A set of subroutines is used to transfer control between the file management root and an overlay.

11.6.3 Flow of Control in File Management.

Figure 11-13 shows the flow of control through the various parts of the file management processor and how a request is handled. Each transition in the figure is numbered. These numbered transitions are described in the list that follows.

```
  +--------+  1  +-------------+          2          +--------+
  | IOPREP |---->|   FMPREP    |-------------------->| FMTCAL |
  +--------+     |             |                     +--------+
                 |             |                        | 3
                 |             |                        V
                 |             |   +--------+   6    +--------+
                 |             |   | FMTSET |<-------| FMACTV |<---+
                 |             |   +--------+<-+     +--------+    .
                 |             |       |       |         * 4       .
                 |    /  \     |       V       |         *         .
     10       YES|   / OP  \   | 9 +--------+  |7        V         .
     +----------- <COMPLETED><-----| FMTRTN |  +-OP PROCESSOR<-----+
     |           |  \ ?  /     |   +--------+<-+         *         .
 UNBUFFERING     |   \  /      |               |         *         .
     |           |     V       |               |         * 5       .
     |           |    11|  NO  |               |         V         .
     V           |      |      |               |8    +--------+    .
 +--------+      |QUEUE REQUEST|               +-----| FMACTV |    .
 | NFTRTN |      | TO FM QUEUE |                     +--------+    .
 +--------+      +-------------+                         .         .
                        |                                . 14      .
                        V       12                       .         .
                    +--------+  +--------+               .         .
                    | IORTN  |  |        V               .         .
                    +--------+  |   +--------+  13       .         .
                        |       |   | FMTASK |---------------------+
                        V       |   |        |           .
                    +--------+  |   |        |           .
                    | NFSCHD |--+   |        |<----------+
                    +--------+      +--------+
                                        |  15
                                      NFEOR
                                        |  16
                                        V
                                      SVC 24

-- XOP Level
.. Task Level
** XOP or Task Level
```

Figure 11-13 Flow of Control in File Management

1. IOPREP buffers the call block, sets the busy flag in the IRB system flags field, and examines the LDT to determine the nature of the request. All extensions to the call block appropriate to the file type and access mode are buffered. Note that if the unblocked I/O flag is on in the LDT, all files appear as relative record files and the call blocks are buffered accordingly. Control is passed to FMPREP by a Branch instruction.

2. FMPREP is contained in the SVCSHD segment, along with IOPREP, and continues running with the caller's JCA mapped into segment two. The context of the caller is saved in the TSB. A check is made to allow initialization on the first call (discussed below). Next, the I/O opcode is examined. If it requires a data buffer, the user's buffer is checked by calling RPSGCK. If the opcode is a read (of various forms), the protection of the buffer is also checked by calling RPPRCK. The file management activity record, the FWA, is initialized with the stack pointer, FWA address, and IRB address. FWAXWP and FWAXPC are set up to transfer control to FMTRTN. A branch to FMTCAL is executed.

3. FMTCAL (module FMTRAN) saves the third segment's limit and bias register values on the current stack, which is the scheduler stack. Then an LWPI instruction is executed to access the transition workspace, which is a partial workspace (contains R10 through R15) in FMTRAN used to manipulate the system map file 0 on the way to the workspace in the FWA. The third segment limit and bias are set to map in the file manager code segment. An RTWP instruction is executed which places execution control at the beginning of FMACTV.

4. FMACTV locates the FCB and RPB and makes some preliminary checks to see that operation can be continued at XOP level. If there is any active operation outstanding for the FCB, or if the current request is active and there is any passive operation outstanding for the FCB, FMACTV queues this request on FCBRLA queue and calls FMTSET. Processing continues at step 6. Otherwise, the opcode is decoded and the appropriate processor called.

5. A processor is included for each one of the file management I/O opcodes. Processing can continue at XOP level if the following three conditions are met:

   * A Write operation must not have the forced write bit set in the LDT

   * The file must be blocked

   * The block must be in memory

   Sequential files may be entirely or partially processed at XOP level. When a required block is not in memory, FMBRD calls FMTSET to transfer to task-level code. When the opcode processor completes, it returns to its caller, which is usually FMACTV. Exceptions are when FMWTF and FMFBSP are called for the Rewrite operation, and when FMFBSP is called for the multirecord Read operation and for Open Extend on sequential files.

6. At several points in processing the call, processing at XOP level can be interrupted. This is denoted by transitions 6 and 7. Whenever this happens, the code call FMTSET, and FMTSET executes a BLWP through FWAXWP, which was set up in step 2 to transfer control to FMTRTN.

7. (See step 8.)

8. After completion at XOP level, FMACTV calls internal subroutine FMNEXT to dequeue appropriate number of requests from FCBRLA queue, and executes a BLWP through FWAXWP, which transfers control to FMTRTN. Processing completes through steps 9 and 10.

9. FMTRTN (module FMTRAN) restores the map file to the point of the call to FMTCAL. It then branches to FMFXRT, an entry point in FMPREP. An incomplete call causes processing to continue at step 11.

10. The operation has been completed at XOP level and exits to IORTN.

11. The operation has not been completed at XOP level. NFQUEH is called to queue the operation for task-level action; to exit, a branch is made to IORTN. IORTN handles initiate I/O, and branches to the scheduler.

12. Eventually, the scheduler chooses the activated file management task for execution. Control is passed to FMTASK, which is the task-level driver for file management requests that must complete at task level.

13. FMTASK begins in a workspace and stack that is unique to the task. It dequeues the first request whose FWAQW is not set from JITFMQ. Using the state information in the FWA, it sets up the environment of the request by performing an SMLOAD call on the caller task segment if a buffer is required. The PC stored in FWAPC by FMPREP is placed in the FMTASK R14. FWAXWP and FWAXPC are set up to transfer control back to FMP210. An RTWP is executed, transferring control to the point in FMTSET after the BLWP, using the workspace and stack allocated with the BRB and begun in step 5.

14. After completing an operation at task level, FMACTV executes a BLWP through FWAXWP, which FMTASK set up in step 13 to cause completion of the operation to return to FMTASK.

15. FMTASK queues the completed request for unbuffering by calling NFEOR.

16. If requests that need to be processed are queued to the request queue, file management continues processing. If JITFMQ has no more requests, an SVC >24 is executed to await the next file management request.

When file management takes end action, a system crash occurs with crash code >A0. This crash indicates that file management encountered an error from which it could not recover.

11.6.4 Overlay Management.

File management overlays are used according to conventions similar to those used for system overlays. File management routines in overlays can be coded in one of two ways.

* By using the conventions for standard subroutines. Register 11 and some other registers are pushed onto a stack upon entry, and these are popped from the stack upon exit. These routines can be called from inside the overlay with FPRCAL. They can be entered from outside the overlay if a word pointing to the routine is defined in the table at the beginning of the overlay.

* By using a convention that allows routines to be entered only from outside the overlay. Nothing is pushed onto a stack, and the exit is a branch to FPORTN. The entry point for these routines must be defined in a word in a table at the beginning of the overlay. The entry point is reached from inside or outside the overlay by a call to FPOCAL.

Both types of routines use FPRCAL to call routines in the resident root; the resident root may be either File Manager or the operating system.

The routines for managing the overlay pool are found in the module FMPMGR. The following routines are included:

FPOCAL - Pooled Overlay Call
Used to enter an overlay either from the root or from a different overlay. Any routine that has its address defined in the table at the beginning of the overlay can be entered with this call.

FPORTN - Returning From an Overlaid Routine
Used to return from a routine in an overlay. It can be entered either directly or indirectly. If the called routine

directly branches to FPORTN, it must return with the stack in the same form as upon entry to the called routine.

FPRCAL - Overlay Pool Routine Caller
Need arises frequently for routines that are overlay resident to call subroutines within the same overlay or within code resident outside the overlay structure. Such routines are called with FPRCAL. All subroutines called from within overlays must be called using either FPOCAL or FPRCAL. When FPRCAL enters a root subroutine, the overlay entry stack frame is built and R11 points to FPORTN. The normal NFPOP exit performed by such routines will cause exit to occur through FPORTN.

FMEXFL - Extend File Processor
One routine in an overlay, FMEXFL, must be exempt from the overlay rules. FMEXFL performs the extend file function. It must be callable from both the File Manager, which uses pooled overlays, and from the task loader, which does not. (The task loader uses FMEXFL to extend the roll file.) Hence, FMEXFL must never call anything that calls another overlay. FMEXFL calls certain root routines, but does so directly because it cannot assume the availability of the pool manager routines. An overlay area is not marked available until the calling task needs another overlay; consequently, the overlay area in which FMEXFL is executing is never marked available as long as it does not call anything that calls an overlay.

An overlay area consists of enough reserved space for the largest overlay plus its relocation bit map, preceded by seven words of overhead, as shown in Figure 11-14. The first three words should be initialized to zero by the assembly and link process that creates the file management subsystem. They are initialized by the overlay load software the first time the overlay area is used. The SIZ field is initialized to the size in bytes of the COD area. OVN is initialized to -1 to flag that no overlay is in the area. USE is initialized to zero. OAD is set up to point to the next overlay area in the pool; OAD points to zero for the last overlay in the pool. COD is the area reserved for code. When an overlay is in use, the field FWAOAD points to OADCOD of the overlay area containing the code.

```
        +-----------------------------------------------+
OADSMT  |        SMT SSB ADDRESS FOR OVERLAY AREA       |
        +-----------------------------------------------+
OADSSB  |        SSB ADDRESS FOR OVERLAY AREA           |
        +-----------------------------------------------+
OADOFF  |        OFFSET OF AREA IN SEGMENT              |
        +-----------------------------------------------+
OADSIZ  |        SIZE OF AREA                           |
        +-----------------------------------------------+
OADOVN  |        OVERLAY NUMBER                         |
        +-----------------------------------------------+
OADUSE  |        USE COUNT                              |
        +-----------------------------------------------+
OADOAD  |        ADDRESS OF NEXT OVERLAY AREA           |
        +-----------------------------------------------+
OADCOD  |                                               |
        /        SPACE FOR CODE OF OVERLAY              /
        |                                               |
        +-----------------------------------------------+
```

Figure 11-14 Overlay Area Structure

Overlays are read into an area obtained from a pool. The pool is a linked list of areas, and each area is a piece of memory large enough for the largest File Manager overlay plus its relocation bit map. When a new overlay is entered (FPOCAL), a piece of STA is obtained for the overlay to use for its run-time stack. Whenever any routine call is made to or from an overlay (FPOCAL or FPRCAL), an overlay return frame is built on the stack.

Overlays are relocated at load time, eliminating the necessity for self-relocating code. The relocation algorithm relocates only those references that are within the address space of the overlay. Relocation requires that all return addresses that reference an overlay be made relative to the beginning of the overlay before being stored on the stack.

When an overlay is needed, a search is made of all areas. If an area is found with the desired overlay, the use count is incremented and the code is entered. If an area is not found, either of two paths are possible. In the first, the overlay is loaded into an area with a zero use count, and the code is entered. The return code links any area whose use count goes to zero to the new end of the list; also, note that the search prefers the oldest available area. Consequently, tasks calling a second overlay tend to keep the first in memory. In the second path, no areas are available (nonzero use count) and the task must queue itself to wait until an overlay area becomes available. Each time an overlay is exited and the use count goes to zero, that overlay area is linked onto the head of the overlay area list. Since the search for an available area proceeds from head to tail of the list, preemption prefers the least recently used overlay areas.

Whenever a call is made to a routine via FPOCAL or FPRCAL, a two-word return frame is built on the caller's stack. The first word pushed on the stack is the return address made relative to the beginning of the overlay. This allows a preempted overlay to be reloaded into a different overlay area, and the return addresses can be rebiased to return to the proper code. The second word pushed on the stack is the overlay index from the contents of R9 when the overlay was originally called. A call from the root produces a stack frame with the return address and -1 for the overlay number. The return logic recognizes the overlay number and marks FWAOAD with zero, indicating no overlay; and FWAOAD activates a waiting task if the overlay area becomes available.

11.6.5 Buffer Management.

The buffer management module (FMBIO) contains routines used to access physical blocks of files. The routines are called from the modules that process blocked files: relative record, sequential, and key indexed. Blocking buffers are mapped into the middle segment, in place of the FMT, making it necessary for file management to copy parts of the FCB that must be accessed while a buffer is being accessed.

FMBRD - Reading a Block from Disk

FMBRD checks the mode of execution. If in XOP mode, SMFSID is called to determine whether the block is in memory. If so, the block is mapped into the current map file 0, and the routine enters NRC000 at label NRC010 to complete processing. If the block is not in memory, FMBRD transfers to task mode and calls FMBSEG.

If FMBRD is executing in task mode to begin with, the memory residence checked is skipped and the routine calls FMBSEG. FMBSEG is called from the entry point NRC000, an entry point that performs common processing for FMBRD and FMBNEW. NRC000 obtains the flags from the OVB used to initialize the File Manager flags in the FMT; these flags are used for buffer management operations.

FMBSEG calls FMCHGS to get the buffer. The buffer is obtained from memory (cached), from disk, or is empty (if routine FMBNEW was called). FMCHGS first tries to get the buffer from memory. If the buffer is not in memory, File Manager is placed on the WOM queue. The task loader is then responsible for loading the buffer from disk or for creating the empty segment.

If execution is in task mode, NRC000 inhibits the scheduler after obtaining the buffer segment. This routine initializes the FWA parameters that describe the block. Those parameters are used in the blocked record transfer.

FMBNEW - Obtaining a Fresh Block
FMBNEW always calls FMTSET at entry to guarantee task-level execution. Then, it enters NRC000 to finish processing for the new block. It also calls an internal entry, FMBCKS, to extend the file, if necessary.

FMBW - Writing a Block to Disk
FMBW always executes in task mode and processes an operation equivalent to a Forced Write Segment SVC.

FMBREL - Releasing a Block After Use
FMBREL returns a block to the cache list ready for future use. If the memory is needed, the block can be released. The modified flag is left in its current state. The JCA is mapped in to replace the block. In XOP level, special code sets the modified and releasable flags in the OVB.

FMBRMD - Releasing a Block After Modification
FMBRMD releases a modified block to the cache list. Marked modified, the block will be written before the memory is released. This routine clears the not modified flag in the File Manager's segment flags and passes execution control to to FMBREL.

FMBWRN - Rename and Write Block
FMBWRN is called by the KIF Manager to write a record to a new record position in a key indexed file. FMBWRN modifies the buffer segment ID, writes the record to the new position, and restores the buffer segment ID to its original value.

11.6.6 Details of I/O Sub-Opcode Processors.

The following paragraphs describe the processing of the operations of Read, Write, Close, Multiple-record Read, and Multiple-record Write for files.

11.6.6.1 Read.

FMRDF handles all requests for disk files (except requests for key indexed files). FMRDF calls FMBKOF to compute the block number and offset and to place the results in the FWA. FMRDF calls CHKEOM to ensure that the desired record is within the range of the file. Next, record locking is checked. FMSETR is called to check for odd buffer addresses and record lengths.

FMSETR places the minimum of the user-specified buffer length and the logical record size of the file into R5, and leaves the user-specified buffer length in R4. The values remain in these registers and are used by the blocked file handlers. Inside the blocked file handlers, the value in R5 is stored in the fourth word of the block specified for relative record files; R4 is stored there for sequential files. Unblocked files are transferred to task level for processing, which occurs in FMRDF.

11.6.6.2 Write.

FMWTF processes all requests for writing to disk files (except requests for key indexed files). FMWTCK is called to determine whether the file is write protected and whether the user has write privileges for the file. FMBKOF is called to compute the block number, offset, and file part needed to satisfy the user's request. Next, the locked record chains are checked. If the record is locked by another LUNO, an error is returned. If the record is locked by the requesting LUNO and the unlock bit in the BRB is on, the record is unlocked. Next, FMSETR is called to set up the user buffer descriptor. In R5 it returns the the file logical record length or the user-specified buffer length, whichever is smaller. It returns R4 as the user-specified buffer length. For relative record files, R5 is used as the record length for writing. For sequential files, R4 is used. When the user's request is shorter than the logical record, the record is zero-filled on the right.

11.6.6.3 Close.

The close function (FMCLOS) ensures that all modified blocks and the FDR are updated on disk.

11.6.6.4 Multiple-Record Read.

Multiple-record I/O is supported to all file types except KIF.

The format for multiple-record I/O is as follows: the length of each individual record is stored in the word immediately preceding the data. Odd-length records are supported; the record is placed in the user buffer with the proceeding length word containing an odd number, but the next record begins on the next word boundary following the odd-length record.

For a read, the user specifies in the read character count the total size in bytes of the space available for data. The system places records in the user's buffer, beginning with the length (in bytes) of the first record, followed by the record, followed by the length (in bytes) of the next record, and so on. Only whole records are transmitted. When the read completes, the output character count specifies the total buffer length in use, including the length words preceding each data record.

If an end-of-file record is encountered and the read buffer contains at least one record, the end-of-file flag is not set and the buffer is returned to the user. The end-of-file flag is set on the following read, and an empty buffer is returned.

Blank adjustment is ignored on multiple-record reads. For example, if the record consists of 20 bytes, 20 bytes are read and the header word contains 20. If the file contains a record with 80 bytes, of which the last 60 are blanks, 80 bytes are read. The padding that occurs when blank adjustment is requested on a normal read is not done on multiple-record reads. All multiple-record read logic is contained in FMMRR.

11.6.6.5 Multiple-Record Write.

The user defines a buffer for a multiple-record write in the same format as created by a multiple-record read. Each record is transmitted to the file in order of increasing memory address. Blank adjustment is honored on each record. Take care to ensure that the character count is accurate. It must include all header words and data. However, if the length of the last record is odd, the character count may include the byte following the proper end of the last record. All multiple-record write logic is contained in FMWRW.

11.6.7 Lower Level Support Routines.

The following paragraphs describe the support routines for concatenated files, unblocked relative record files, and blocked files.

11.6.7.1 Concatenated Files and Multifile Sets.

Concatenated file and multifile set handling involves predominantly three routines: FMBKOF (for nonkeyed files), KMBEG (for key indexed files), and FMBCLO (for both key indexed and nonkeyed files).

Unblocked Relative Record Files.

FMBKOF is called to map the logical record number to a block number and offset. For unblocked files, this consists of determining which part of the concatenation has the specified record. The logical record is checked against the allocations of each file in the concatenation list. The correct FCB address is placed in the FWA. The record number is biased by the allocations of the preceding files, and the result is stored in the FWA for use in direct disk I/O.

Blocked Relative Record Files.

FMBKOF checks the logical record number against the allocations of all files in the chain, and the FDP of the correct file is placed in the FWA. The logical record number is then biased by subtracting the allocations of previous files, and the result is used with the physical record length of the individual file to compute the block number and offset of the specified record. The block number and offset are placed in the FWA.

Sequential Files.

FMBKOF uses the currency information in the RPB. The FCBs are scanned to find the file containing the current record. The appropriate FCB address is placed in the FWA, and the block number and offset are copied from the RPB to the FWA.

Multifile KIF Sets.

Multifile KIF sets are handled in KMBEG, in the KMRD routine. The block number desired is checked against the allocations of each of the files, and the FCB address of the correct one is placed in the FWA. The block number needed is biased by the allocations of the previous files to obtain a relative block number.

Closing Blocked Files.

To close a concatenated file, IOPREP closes the assigned LUNO. Then, file management completes the Close operation. FMCLOS calls FMBCLO to obtain all modified buffers written to the disk. FMBCLO calls SMFLSH and loops through each part of the concatenation to obtain all modified blocks written.

11.6.7.2 Unblocked Relative Record Files.

Records for unblocked relative record files are transferred directly to and from the user buffer. Each record begins on a sector boundary and occupies contiguous disk for the specified length of the logical record. File management must queue requests for unblocked files through task level.

11.6.7.3 Blocked Files.

Records for blocked files are transferred through an intermediate buffer allocated from free memory. The physical block on disk begins on a sector boundary and occupies contiguous disk for the specified length of the physical record. If the block size is larger than an ADU, only an integral number of blocks are placed in an ADU, and the first block must begin on the ADU boundary.

Record Transfers.

The transfer of the record from blocking buffer to user address space is handled in map 0 by routines called through NFMAPO. The routines used are NFCXFR, FMBSRD, and FMBSWT. The first one is used for relative record and unsuppressed sequential files. The other two are used for blank suppressed sequential files. FMBSRD unsuppresses from blocking buffer to user buffer, and FMBSWT suppresses from user buffer to blocking buffer. These routines are called by FMCPB, a routine in the root entered by the NFMAPO call. One of the parameters of the call is the address of the transfer routine (one of the three routines listed above). The calling sequence to FMCPB is set up by routines in FMCPBI, a module containing the routines FMCIRD and FMCIWT. These routines use the information in the FWA, the user buffer descriptor, and the blocking buffer parameters to set up the call to FMCPB.

Relative Record Files.

FMRDBR is the blocked relative record handler. It calls FMBRD to obtain the block. If the needed block is not in memory, FMBRD calls FMTSET to obtain the block. The block, offset, and user buffer descriptor are then used in FMCIRD (module FMCPBI) to call FMCPB, which transfers the record. Blocked relative record files store data by the following algorithm. The block number is determined by dividing the logical record number requested by the blocking factor. The quotient is the block number. The remainder is multiplied by the logical record size of the file to determine the byte offset into the block at which the record is found.

Sequential Files.

FMRDSQ is the sequential file read handler. It calls FMBRD to obtain the blocks needed to process the request. If the block is not in memory, FMBRD calls FMBSEG to execute the Change Segment SVC. As for relative record files, the block, offset and user buffer descriptor are passed to FMCPB via FMCIRD to transfer the record. For blank suppressed files, the address of the appropriate routine is supplied in the calling sequence.

Blank Adjustment.

Blank adjustment on read is performed after the record is transferred to the user's buffer. Space left unfilled in the user's buffer after the read is filled with blanks. This function is selected by the blank adjust bit in the user's IRB. It is performed by FMRDSQ.

Blank adjustment on write is performed before the record is transferred to the blocking buffer. The length specified to be written is used to find the end of the record in the user's buffer. Trailing blanks are counted, and only the characters up to and including the last nonblank character are transferred to the blocking buffer. This function is performed by FMWTSQ.

11.7 KIF MANAGEMENT

A set of routines used only for key indexed files is supplied. These routines, each having KM as the first two characters of its name, should be thought of as a subsystem of file management. The buffer management routines of file management are used to perform low-level I/O functions. Eight system overlays are used by the

KIF manager to perform specific functions.

When a user issues a KIF request, control is passed to IOPREP from RPROOT, the SVC processor. IOPREP performs standard preprocessing and passes control to IOCHKX to finish preprocessing. In addition to performing the same functions as for other files, IOCHKX performs special processing for KIF. The key number is validated to ensure that it is legal for the file. Space for an IRB and KIF currency block (KCB) is allocated. If currency is required, that is, if the sub-opcode is greater than >40, IOCHKX validates that the currency block is contained in one map segment and that the segment is not write protected. The currency block is then buffered into the STA, and control is passed to FMPREP for further preprocessing.

After determining the size of the largest key, FMPREP allocates enough space for a KIT, which contains a FWA, plus three times the sum of the largest key and six. The extra six bytes for each of the three keys is for overhead. Next, if a currency block is present and the currency block contains a valid key pointer, FMPREP validates that the key is contained in one map segment and copies the key into the first of the three key buffers at the end of the KIT. IRBFWA is set to point to the KIT. FMPREP then branches to FMTCAL, which causes FMACTV to be entered using the workspace in the KIT

FMACTV transfers control directly to KMBEG, the main driver for the KIF subsystem. Control is passed to KMBEG with the address of the buffered request in register 12 (R12). The format of the complete buffered request is illustrated in Figure 11-15.

```
+--------+    +->+--------------------------------------------+
|  BRO   |    |  /          WORKSPACE FOR KIF          (FWA)   /
+--------+    |  +--------------------------------------------+
|   :    |    |  /          WORKING STORAGE FOR KIF    (FWA)   /
+--------+    |  +--------------------------------------------+
|IRBFWA  |----+  /          STACK SPACE FOR KIF        (FWA)   /
+--------+       +--------------------------------------------+
|   :    |       /          MORE WORKING STORAGE FOR KIF (KIT) /
+--------+       |          (INCLUDES AREA FOR THREE BUFFERED  |
|  KCB   |       |          KEYS AND OVERHEAD)                 |
+--------+       +--------------------------------------------+
```

Figure 11-15 Buffered KIF Request

The FWA and KIT are always referenced using R15 as a base register; the BRO, IRB, and KCB are referenced using R12.

11.7.1 KIF Data Structures.

KIF routines use various data structures in addition to the structures used by the File Manager (that is, FCB, FDR, and FWA).

These structures are the B-Tree Block (BTB), KIF Currency Block (KCB), KIF Information Block (KIB), and KIF Task Area (KIT). Each of these is shown in detail in the section on data structure pictures.

B-Tree Block (BTB)
The BTB is a disk-resident structure that contains the overhead for a key file. When a data record must be located, each level of the B-tree associated with the key is read in tree order to locate the record. KIF maps the BTB into its second map segment to access it.

KIF Currency Block(KCB)
The KCB is used by KIF to maintain currency for the user from operation to operation. When an operation is performed on a key file, the user's currency block is buffered into the STA along with the IRB. The KCB is shown in detail in Figure 11-16.

The first four bytes of the KCB are defined in the DNOS SVC Reference Manual. Bytes 4 through 9 (a three-word entry) are the data block key for the data record. The first two words contain the block number for the current data record. The last word is the logical record ID for the current record. Bytes >A through >F are the B-tree pointer. The first two words contain the block number for the current B-tree entry. The last word is the address of the current B-tree entry when it is mapped into the KIF task (that is, the address when mapped into the second segment of the task). Bytes >10 and >11 are the size (in bytes) of this B-tree entry (that is, the size of the current key plus six bytes of overhead, rounded up to an even number). Byte >12 is used to store the last opcode used (for those operations that perform different functions depending on the last operation).

KIF Information Block (KIB)
The KIB is a disk-resident data structure. It contains data records for the file.

KIF Task Area (KIT)
The KIT is used by KIF as additional working storage. First, the KIT contains a FWA, which contains a workspace, stack and other data. Next, several fields of the FCB are buffered so that the FCB need not be mapped in when these fields are required. These fields are the file extent (or allocation) (FCBEXT), the block end-of-medium (FCBBKM), and the KIF extension to the FCB (includes current command number, current log block, free block queue head, and the B-tree roots). Next, the KIT has pointers to the three key buffers that reside at the end of the KIT, the key number and size of the key being processed, and a B-tree stack used to contain the block number and B-tree entry address for each level of a B-tree. (This address is built when transversing down each

level of a B-tree searching for an entry. This allows tracing back up the B-tree to modify a higher-level entry.) Next, the KIT contains temporary storage for the KIF routines. Finally, the three key buffers reside at the end of the KIT. Each buffer contains room for the longest key plus six bytes of overhead.

```
Dec.
Byte
----- *---------------------------------------------------*
  0   |    INFORMATIVE CODE     |      KEY NUMBER         |
      *---------------------------------------------------*
  2   |                KEY VALUE POINTER                  |
      *---------------------------------------------------*
  4   |            DATA BLOCK RECORD NUMBER               |--- DATA
      |                                                   |   |BLOCK
      *---------------------------------------------------*   |KEY
  8   |            LOGICAL RECORD ID VALUE                |---
      *---------------------------------------------------*
 10   |         B-TREE LEAF NODE RECORD NUMBER            |---
      |                                                   |   |B-TREE
      *---------------------------------------------------*   |POINTER
 14   |  MEMORY POSITION OF KEY VALUE IN B-TREE ENTRY     |---
      *---------------------------------------------------*
 16   |                 B-TREE ENTRY SIZE                 |
      *---------------------------------------------------*
 18   | LAST OPCODE USED        |  CURRENT OPCODE         |
      *---------------------------------------------------*
```

Figure 11-16 KIF Currency Block

Bytes (Dec)   Field Use

User-Supplied

0   Used as an input value when the partial key feature is used. Used for set currency operations, read greater, and read greater or equal. Used as an output field for return of informative codes under all circumstances

1   For a key dependent operation, used as an input value to specify the number of the key to be used, ranging from 1 through 14

2-3   Address of the buffer containing the key to be used in a key dependent operation

System-Defined

4-7   Two-word number giving the physical record number of the physical record that holds the logical record associated with the currency

8-9 A number, unique within the physical record, associated with the logical record within the physical record that is associated with the currency

10-13 Two-word number of the physical record of the key indexed file that holds the leaf node with the key associated with the currency

14-15 Address of the currency related B-tree entry when the physical record containing the leaf node is mapped into the key indexed file manager address space. A B-tree entry is composed of a two-word physical record number, a logical record ID, and a key value

16-17 The length of the key plus six

18 Opcode of the last operation performed using this currency block. Read next and read previous operations check this value and if it is equal to the set currency opcode, they perform a read current. Read previous also checks for the delete opcode, in which case a read current is also performed. Other operations also use this field

19 Current opcode

11.7.2 KIF Management Code Structure.

KMBEG is the main driver for key indexed file operations. It decodes the operation code and passes control to the processor for the specified operation. Table 11-4 lists the main processor for each operation.

Table 11-4 KIF Main Routines

Name	Operation Performed
KMCLOS	Close (overlay residing in KMOPCL module)
KMDEL	Delete by Key and Delete by Current (overlay residing in KMDLSR module)
KMINSR	Insert (overlay)
KMOPEN	Open Random (overlay in KMOPCL module)
KMRN	Read Next
KMRP	Read Previous (contained in KMRN module)
KMRR	Read by Key, Read Current, Read by Primary Key
KMRSQ	Forward Space, Backspace, Read ASCII, Rewind (overlay in KMOPCL module)
KMRW	Rewrite (overlay)
KMSC	Set Currency Equal, Equal or Greater, and Greater (overlay residing in KMDLSR module)

The Read Greater and Read Greater or Equal operations are performed by first calling KMSC, then KMRRC (entry point in KMRR). Several subroutines are used by KIF in addition to the buffer management routines of File Manager. These are listed in Table 11-5.

Table 11-5 KIF Subroutines

Name	Function Performed
KMBDEL	Modify higher level BTB (overlay)
KMBIN	Search for key value in given BTB
KMBS	See KMBTS (entry point in KMBTS)
KMBSC	Compute blank suppressed size
KMBTD	Delete B-tree entry
KMBTI	Insert entry into specified B-tree
KMBTIS	Perform B-tree split (overlay)
KMBTS	Search a key's B-tree for match on key value
KMCNV	Convert character strings by country code
KMEK	Extract key from blank-suppressed file and unblank
KMGEB	Get empty block for B-tree splits (entry point in KMGF)
KMGFB	Get free block for record insert or rewrite (entry point in KMGF)
KMKC	Compare two character strings of given size
KMKDG	Find key descriptor entry, given key number
KMLOC	Locate B-tree entry, given user currency
KMLOG	Log current block
KMPLG	Recover from an insert error while in partial logging mode
KMRDK	Read in data record by key
KMRWS0	Delete old key and insert new key (overlay in KMRWS module)
KMRWS1	Get new block for record (overlay in KMRWS module)
KMRWS2	Map in new block and write out old (overlay in KMRWS module)
KMRWS3	Update B-trees for new record position (overlay in KMRWS module)
KMRF	Return block to free chain (entry point in KMGF)
KMSUK	Compute key size and B-tree entry size
KMTAB	Character conversion tables
KMULG	Unlog blocks to their position in file
KMWRN	Force write a segment to a specified address

11.7.3 Details of KIF Operations.

Once File Manager has determined that an operation is being performed on a key indexed file, control is passed to KMBEG. KMBEG buffers certain fields of the FCB into the KIT, tests validity of input parameters, initializes the save area in the KIT, and passes control to the processor for the specified operation. If an error is encountered in the operation processor, control is returned to KMERR (entry point in KMBEG), which calls KMULG to unlog any records that have been logged, skips the unbuffering of the KIT into the FCB and returns control to the caller (with an error). If the operation completes with no errors, control is returned to KMCDN (entry point in KMBEG), which unbuffers the KIT into the FCB and updates the FDR if the free chain has been modified and returns to the caller.

11.7.3.1 Close.

The Close operation is performed by KMCLOS. Record 0 is updated by setting the log command number to 0 so that no unlogging can occur on the subsequent open. Then, FMCLOS is called to perform the close processing for a file. When control returns from FMCLOS, the Close operation is complete.

11.7.3.2 Open Random.

The Open operation is performed by KMOPEN. FMOPEN is called to open the file. Upon return from FMOPEN, KMOPEN calls KMULG if there are no other LUNOs open to the file. FMOPEN does not use the currency block.

11.7.3.3 Read Greater and Read Greater or Equal.

These operations are handled by two separate processors. First, KMSC is called to establish the currency for the subsequent read operation. KMSC calls KMBTS to search the B-tree of the specified key for a matching key value; if no match is found, KMBTS searches the first B-tree entry with a key of greater value. If no match exists, an informative code is returned to the caller. Otherwise, KMSC returns to KMBEG, which calls KMRRC to process the Read operation. If the file is single keyed, KMLOC is called to read the B-tree record described by the currency set up by KMSC. This is required since the key value is blanked out in the data block and therefore must be obtained from the B-tree block.

Next, KMRRC calls KMRDK to read the data block to which the currency block points. The data block is moved to the user's buffer via the FMCIRD routine. If the file is single keyed, the key value is moved into the user's record. Finally, record locking, if specified, is accomplished by the FMLKCK and FMLKON routines.

11.7.3.4 Read by Key, Read Current, and Read by Primary Key.

The Read Current operation is processed the same as the Read Greater operation except that the KMSC step is eliminated (since the currency is already set.) The Read by Key and Read by Primary Key operations are also processed like the Read Greater operation except that the currency must be set before control is turned over to KMRRC. The currency is set in KMRR by calling KMBTS to search the B-trees to find the block holding the specified key. If the block is found, the currency is established and control is passed to the KMRRC routine.

11.7.3.5 Read Next.

The Read Next operation is performed by KMRN. KMLOC is called to locate the next entry from the current entry set up by a previous operation. If such an entry is found, the Read Random (KMRR) code is entered at entry point KMRRB. KMRRB reads the data block, moves the record to the user's buffer, and moves in the key value (if the file is single keyed).

11.7.3.6 Read Previous.

The Read Previous operation is performed by KMRP (entry point in KMRN module). This operation is identical to the Read Next operation except that KMLOC is called to locate the entry that precedes the current one.

11.7.3.7 Insert.

The Insert operation is performed by KMINSR. KMINSR first checks to see if the file is single keyed. If so, the key value is blanked in the user's buffer (Note: The key value is not duplicated in the data block, thereby freeing disk space.) KMBSC is called to compute the blank-suppressed size of the user's record. If the size is greater than the physical record size of the file, an error is returned to the caller. (Before the error is returned, the key is restored to the caller if the file is single keyed.) KMGFB is called to find a free block with enough space to hold the blank-suppressed data record. The record number of this block is stored in the currency block, and the user's data record is transferred to it via the FMCIWT routine. Once completed, the data record is written and the key value is restored to the user's buffer.

Now that the data record is in the data block, an insert into the B-tree is required for each key of the record. KMBTS is called to search the B-tree of the key for a matching key value. If no match is found, KMBTS searches for the first entry with a greater key value. If a match exists, a check is made to see if duplicates are allowed on the key. If not, an informative code is returned to the user. Next, KMBTI is called to insert the entry into the B-tree for the key. This sequence is followed for each key. The end-of-medium record number is incremented, and a flag is set to inform KMBEG that the FDR must be updated.

When the partial logging bit is set in the primary key flag word of the KDB, KMINSR force writes the data block and defers the writing of all other blocks. If an error occurs after the first key insert and before the last, an error recovery overlay is loaded. The error recovery overlay (KMPLG) uses KMBTD to delete the data record from the data block and deletes the keys which were just inserted. This allows the capability of unlogging an insert while taking advantage of buffer caching to increase the speed of an insert.

11.7.3.8 Rewrite.

The Rewrite operation is performed by KMRW. First, KMRW checks to see if the file is single keyed; if so, KMRW blanks out the key in the user's buffer. (Refer to the description of this function in the paragraph on Insert.) A check is made to ensure that the record size is less than the physical record size of the file. The key is then replaced in the user's buffer (if single keyed). Next, if the file is single keyed, KMLOC is called to obtain the key value for the key specified by currency. KMRDK is called to read the block that contains the old record, and a check is made to determine if the new record will fit in the old record. If not, the old record is prelogged by KMLOG, and the overlay KMRWS1 is called to obtain a new block to hold the record.

KMEK is called to copy the old key into the second key buffer (in KIT), and the user's key is copied into the first key buffer. KMKC is called to compare the values of both keys. If the values are different, a check is made to ensure that the key is modifiable. If not, an error is returned to the caller. However, if the key value is different and the key is modifiable, the KMRWS0 overlay is called to delete the old key value and insert the new. If more keys exist for the file, the key values are updated for each one.

Once the key values are updated, the new record must be placed into the appropriate block. This is accomplished by first seeing if the new record is the same size as the old record. If not, the old record space is removed from its block. If the new record is smaller than the old record, it is inserted into the old block. If the new record is larger, the KMRWS2 overlay is called to write out the old block and map in the new block, which was retrieved earlier in a call to KMRWS1. Finally, the new record is moved into the block by FMCIWT. (If the file is single keyed, the key value is blanked out.) The data block is written; if a new block was used for the data record, the KMRWS3 overlay is called to update the B-tree for the new location. If unlocking was specified, FMLKOF is called to turn off locking on the record.

11.7.3.9 Delete by Key and Delete by Current.

The Delete by Key and Delete by Current operations are performed by KMDEL. If Delete by Key was requested, the currency is set up by a call to KMBTS. KMLOC is called to read the BTB described by the user's currency in order to locate the data record for the key. A check is made to ensure that the record being deleted is not locked. KMRDK is called to read the data record described by currency. The key value is copied into a buffer (by KMEK). For each key of the file, KMBTD is called to perform the delete. After the key is deleted, the data record is deleted. This is achieved by first prelogging the data block, then moving up each entry below the entry to be deleted. The block is then linked to the free chain, if it is not already there, by calling KMRF. The data block is written to its file position, and currency is modified to point to the preceded entry in the data block. The end-of-medium record number is decremented, and a flag is set to inform KMBEG that the FDR must be updated.

11.7.3.10 Set Currency Equal, Greater, Equal or Greater.

The Set Currency Equal, Set Currency Greater, and Set Currency Equal or Greater is performed by the KMDLSR overlay. If a Set Currency Greater is executed, a value of 1 is added to the last byte of the key in order to get the next greater entry. KMBTS is called to perform the B-tree search for the specified key. If a unique match is found, the B-tree and data block currency are set to point to the specified key. If a match is found with duplicates, an informative code is returned to the caller and currency is set up. If a match is not found and a Set Currency Equal was executed, an informative error is returned without currency being set. Otherwise, currency is set to the next greater entry than the specified key. Upon exit, a check is made to determine whether the record is locked; if it is, an informative code is returned.

11.7.3.11 Forward Space, Backspace, Read ASCII, Rewind.

The Forward Space, Backspace, read ASCII, and Rewind operations are performed in KMRSQ. Each of these operations stores currency into the last five words of the RPB. To make six words of currency fit into five words, the first byte of the data base key and the B-tree pointer physical record numbers is not stored. This fact causes no trouble, however, since a file cannot be large enough to use these bytes. If a Rewind operation is performed, the currency in the RPB is simply zeroed. KMRSQ executes Forward Space and Backspace operations by making repeated calls to KMLOC, which locates either the immediately preceding or the immediately following record and sets the KCB to the currency of the record found. KMRSQ calls KMLOC the number of times specified by the IRBOCC field of the call block. After the last call, it copies the currency in the KCB to the RPB. For a Read ASCII operation, KMRSQ calls KMLOC one time to locate the next record and moves the currency from the KCB to the RPB. KMBEG will then call KMRRC to actually read the record.

11.7.4 Details of KIF Subroutines.

KMBDEL

This routine is an overlay called by KMBTD whenever a B-tree entry of the greatest value is deleted. The next-higher-level B-tree must be modified as follows to reflect the new greatest value. First, the new highest key value is copied into the first key buffer for later use. The lower-level block is written back to the file, with the B-tree entry of greatest value deleted. If this was the last entry in the block, it is returned to the free chain by calling KMRF. When the block is returned to the free chain, the predecessor and successor must be linked. Once the lower-level block is released, the higher-level block is read. Before modifying the higher-level block, the block is prelogged by calling KMLOG. Then, if the lower-level block is not empty, the new greatest key value is moved into the higher-level block. If the new key value is also the greatest key value in the higher-level block, the next-higher-level B-tree must also be modified. If the lower-level block is empty, the B-tree entry for the block is deleted; KMBDEL is reentered if either the greatest entry or the last entry was removed. Upon exit the higher-level block is written back to the file.

KMBIN

KMBIN is called to perform a binary search for a given key value within a given B-tree block. This routine receives a pointer to the key value. The B-tree block which is to be searched is mapped in, and R5 is set to point to the B-tree block. KMKC is called to determine in which half of the partition the key resides. The partition is halved until it converges, and KMKC is called to set up the comparison results for output.

KMBSC

KMBSC is called to compute the number of characters that a field would have if it were blank suppressed. A pointer to the field and the number of characters is input. This routine simply computes the number of nonblank characters and includes the blank-suppression overhead word where needed.

KMBTD

KMBTD is called to delete a B-tree entry. First, KMBS (an entry point in KMBTS) is called to search the B-tree with the given key value. If the entry is not found, KMBTD is exited. Otherwise, the B-tree record is read, and the entry is located in the block. The BTB is prelogged by KMLOG; if this key is the one specified by the currency block, the currency information is updated to print the preceding B-tree entry. The entry is deleted by moving up all entries below it in the

BTB. If the block is now empty and the key was the one specified by currency, currency is set up to point to the predecessor BTB. If the greatest key in the block was deleted, the KMBDEL overlay is invoked to modify the higher-level B-tree structure.

KMBTI

KMBTI is called to insert an entry into the B-tree associated with the key specified by the caller. The specified BTB is read and its image is prelogged by KMLOG. If the BTB cannot accommodate the size of the new entry, the block must be split by the KMBTIS overlay. Once a block large enough for the new entry is received, the entry is placed in the block. It may be necessary to move some of the entries in the block down in order to insert the new entry in its required location. The B-tree overhead is updated, and the block is written to its file position.

KMBTIS

KMBTIS is called to perform a B-tree split for KMBTI. A descriptor of the B-tree to be split is input. Figure 11-17 and Figure 11-18 illustrate how the B-tree looks before and after a split. Note that in Figure 11-17 the root is splitting, whereas in Figure 11-18 a regular B-tree node is splitting.

```
Before Split          *-----------*
                      |   root    |
                      *-----------*

After Root Splits
                      *-----------*
                      |   root    | (modified)
                      *-----------*
                            |
                ------------------------
                |new                    |new
           *---------*             *---------*
    (left  | B-tree  |  --------->  | B-tree  |(right
     block)| entries |  <--------   | entries | block)
           *---------*             *---------*
```

Figure 11-17 Example of Root Node Split

```
Before Split             *----------*
                         |   root   |
                         *----------*
                              |
                   ------------------------
                   |A                     |B
              *---------*            *---------*
              | B-tree  |            | B-tree  |
              | entries |            | entries |
              *---------*            *---------*

After Right
(Block B)                *----------*
BTB Splits               |   root   |
                         *----------*
                              |
                 ------------------------------
                 |A(modified)  |new           |B(modified)
            *---------*   *---------*    *---------*
            | B-tree  |-> | B-tree  |->  | B-tree  |
            | entries | <-| entries |  <-| entries |
            *---------*   *---------*    *---------*
```

Figure 11-18 Example of Regular B-Tree Node Split

First, an empty block is retrieved by calling KMGEB (an entry point in KMGF). This block is used for the left BTB. If the B-tree root (lowest-level BTB) is being split, another empty block is retrieved to hold the right BTB. The block that needs to be split is read, and the last entry is copied into the second key buffer. The new entry is inserted into the data block, which may require moving down other entries. Next, a check is made to determine whether the block should be split 50/50 or 90/10. If records are inserted sequentially into the B-tree, the new left block contains 90 percent of the entries, and the new right block contains 10 percent of the entries. Thus, if the user continues to insert sequentially, the next B-tree split will be delayed.

Depending on the split ratio, the number of entries to use in the left block is calculated and the block overhead is updated to reflect the number of B-tree entries and the amount of free space. The successor and predecessor pointers are set up, the new block number is updated, and the new left block is written to the file. The greatest key value in the left block is saved in the first key buffer for later insertion into the higher-level BTB.

Next, the right block is set up by determining the number of entries in it, moving the B-tree entries up to the top of the block, and restoring the last (greatest) entry to the block (saved in the first key buffer at the start). The pointer fields are set up, and entries are defined for the space remaining and the number of entries in use. If the root is split, the new right block number is stored in the block and the block is written to its new file position. A new root block is created, and the two new greatest entries are stored in the root. If the root is not being split, the right block is written to its original file position and the original block's predecessor block has its successor pointer modified to point to the new left block (via the FXPPTR subroutine in KMBTIS).

KMBTS

KMBTS is called to search a given key's B-tree for a matching key value. If no match is found, KMBTS searches the first B-tree entry with a key value greater than that specified. This routine also builds a B-tree stack (saved in KIT) that contains three words (block number and index) for each B-tree level. Each level of the key's B-tree is read, beginning with the root. KMBIN is called to find the matching key value or the next greater value for each B-tree. If a match is found, a check is made for a duplicate entry. If a duplicate entry is found, an output flag is set accordingly. If duplicates exist, the B-tree stack will be set up to point to the last duplicate. The B-tree is read and the B-tree stack is updated for each level until the leaf node is reached. If the specified key value matches the last entry of a block, the successor block is read to determine if a duplicate exists in it. If a duplicate does exist, the above process is repeated until the last duplicate is found (many successors may be read). The B-tree stack is set to the record that contains the last duplicate. Once the leaf node is reached, the output flags are initialized for duplicates and unique match.

KMCNV

KMCNV is called to convert strings, using the country conversion tables set up by KMBEG.

KMEK

KMEK is called to extract a key from a blank-suppressed file, unblank it, and move it to a buffer specified upon input. First, KMKDG is called to find the key descriptor for the specified key. The key size and offset into the record can be determined from the key descriptor. If the file is single keyed, the key value is taken from the first key buffer (which was set up by File Manager) and KMEK is exited. For multi-key files, the key must be unblanked by KMEK rather than by File Manager.

KMGEB

KMGEB is called to obtain an empty block for B-tree splits. The file end-of-medium is advanced, and FMCKEX is called to check the file extension. If necessary, the file will be extended.

KMGFB

KMGFB is called to obtain a free block with adequate space to accommodate a record that is being inserted or rewritten. First, KMGFB checks the free chain pointer in the KIT to determine if there are any free blocks. If not, the end-of-medium of the file is incremented and a new block is read in. The overhead of the block is initialized (block number, space remaining, and free chain pointer), and this block is returned. If free blocks are available, the first one is read, prelogged by KMLOG, and a check is made to determine if it has enough free space to accommodate the record. If the space is sufficient, this block is returned to the caller. If the space is insufficient, the block is removed from the free chain (the free block pointer in the block is set to the value negative 1) and written to the file. The next free chain block is then tried. If a large enough block of free space has not been found after three tries, the new block is taken from the end-of-medium.

KMKC

KMKC is called to perform a logical comparison on two character strings. If the two strings are ASCII, the result corresponds to the ASCII sort order. If the strings are of different lengths, the longer string is truncated and ignored. KMCNV is called to perform any conversion on international text.

KMKDG

KMKDG is called to find the key descriptor entry (located in KIT) with a given key number.

KMLOC

KMLOC is called to read a B-tree record and locate the unique entry described by currency. The caller may specify the current entry, the previous-to-current entry, or the successor to the current entry. KMLOC starts by reading the B-tree record contained in the currency block. The address of the B-tree entry described by currency is located in the currency block (the third word of the B-tree pointer). If this address is zero, either of two situations can result. If the previous-to-current entry is needed, the predecessor block is read to find the entry. If no predecessor exists, an informative code is returned. If the successor to the current entry or the current entry is desired, the first entry in the BTB is used. When the address of the current B-tree entry is non-zero, a check is made to see if this B-tree address points to the data block in the currency. If not, the correct B-tree entry is found by searching through the B-tree. Once the B-tree entry that points to the correct data block is found, the currency is modified according to what was requested (previous-to-current, current, or successor-to-current). When a single-keyed file is used, the key value is copied into the first key buffer.

KMLOG

KMLOG is called to prelog the block currently mapped by the caller. The position in the log records to which the block should be written is kept in the KIT (KITCLB). Also, the log command number for the logging is kept in the KIT (KITCMD). Blocks will not be logged if the force write flag is turned off (that is, a Modify Key File Logging (MKL) command was executed).

KMPLG

KMPLG is called from the insert overlay (KMINSR) to recover from selected insert errors while a file is set to partial logging. KMPLG recovers from insert errors by simulating the roll back action of KMULG. Any keys inserted by the current insert operation are deleted using KMBTD. The data record from the current insert is deleted from the data block. No errors are returned by KMPLG.

KMRDK

KMRDK is called to read a data record with a given data base key. The data base key consists of a block number and key ID. The record is read in an attempt to locate the specified key ID. If the ID is found, the address of the entry is returned. If it is not found, a zero is returned for the address.

KMRWS0

KMRWS0 is called by the Rewrite processor (KMRW) to delete an old key and insert a new key. The old key is passed in the second key buffer, while the new key is in the first key buffer. First, a check is made to see if an old key exists. If a >FF resides in the first byte of the key or the key is blank, the key does not exist. If an old key does exist, it is deleted by the KMBTD routine.

The new key is now inserted into the B-tree. Again, if a >FF resides in the first byte of the key or the key is blanked, the insert is not performed. If a new key exists, KMBTS is called to search the B-tree for the specified key. If a matching key value is found, a check is made to ensure that duplicates are allowed. KMBTI is called to insert the new key into the B-tree. If this key was the currency key, the user's currency is set up to point to the new insert B-tree entry.

KMRWS1

KMRWS1 is called by the Rewrite processor (KMRW) to obtain a new block for a data record. KMGFB is called to return a free block. A new key ID is generated for the record, given the current maximum key ID used. Also, the address where the new record will reside is computed.

KMRWS2

KMRWS2 is called by the Rewrite processor (KMRW) to map in a new block and write out the old block. This routine writes the block currently mapped, reads the new data block, increments the highest key ID used in the block, and sets up the currency for the new block.

KMRWS3

KMRWS3 is called by the Rewrite processor (KMRW) to update B-trees when a data record moves. For each key of the file, the key value is located (if it exists), and the BTBs are searched by KMBTS to find the entry for the key. The block returned by KMBTS is read and searched for the location of the key. When the key that has the same data base ID (that is, pointer to data record) as the old record is found, the new data base ID is stored in the B-tree entry.

KMRF

KMRF is called to return a block to the free chain. The block is simply linked to the free chain and written back to the file.

KMSUK

KMSUK is called to compute the key size and B-tree entry size given the key number. This routine passes a key number to KMKDG, which returns the key descriptor entry containing the key size. The B-tree entry size is the key size plus six.

KMTAB

KMTAB contains conversion tables used to convert standard ASCII characters into a nonstandard collating sequence.

KMULG

KMULG is called to write preimages logged at the beginning of the file (in the preimage log area) to their true positions in the file. This routine is called whenever an Open is executed (to clean up operations only partially completed) and whenever an error occurs in an operation. If the Open routine calls KMULG, unlogging begins at record 0. If the command number in the block is zero, unlogging does not occur.

KMWRN

KMWRN is called to force write a buffer segment to a specified address. This routine temporarily modifies the installed ID in the SSB of the buffer segment mapped into position two and sets the modified flag. FMBW is then called. After the buffer has been written to the file, the SSB is restored to its original state.

SECTION 12

DNOS SYSTEM TASKS

12.1 SYSTEM TASK ENVIRONMENT AND CONVENTIONS

A system task executes with the following segments mapped in by map file 1: the system root, the JCA of the executing job, and the task code. Tasks that run under the system job have the system JCA mapped in. System tasks may need to map out the system JCA and map in another table area using the set of nucleus routines described in a previous section.

A system task that runs in a user job has the user JCA mapped in as its second segment of map file 1. If the task is a system queue server, it has a queue header in the user JCA. RPROOT examines flags in the request definition block (RDB) for an SVC to determine whether or not the server is to be bid in the user's job. If so, RPROOT calls NFQUEH to queue the request to the appropriate queue header in the user's JCA and to bid the server task in the user's job. The queue server is bid with its second parameter being the queue header address. Using this address, the queue server can process its requests and terminate when the queue is empty.

All routines in the system root can be directly accessed by system tasks. In addition, a system task can transfer into map file 0 to use routines in map file 0 or to do work that cannot be done in map file 1. The nucleus function NFMAP0 is used to access map file 0 code; NFMAP0 is described in the section on nucleus functions.

A system task has access to data structures as well as common operating system routines in the system root. The available structures include queue headers, global pointers, global data, and any other structure located in the system table area.

12.2 WRITING AND LINKING AN ASSEMBLY LANGUAGE TASK

System tasks should follow the conventions used for DNOS code (see the section on naming and coding conventions).

The link of a system task needs to include the system root. The link control file shown in Figure 12-1 shows an example, where the object code for the task is in the file PROG.TASK.OBJECT.

VOLOBJ is the volume name of the disk on which the linkable DNOS code resides (that is, the response to the DATA DISK prompted during system generation).

```
NOPAGE
ERROR
PROCEDURE DUMROOT
DUMMY
INCLUDE     VOLOBJ.S$SGU$.DUMROOT
PHASE 0, NEWTASK,PROG >C000
INCLUDE PROG.TASK.OBJECT
END
```

Figure 12-1 Example of Link Control for System Task

12.3 USING OVERLAYS IN ASSEMBLY LANGUAGE SYSTEM TASKS

System overlays are overlays of system tasks that have an entry in a table built during system generation to enable loading in one disk access. They are used in the following subsystems: file management, key indexed file management, disk management, the I/O Utility, and error processing for program management. Such tasks must reside on the kernel program file and sysgen must be aware of the overlays in order to build appropriate linkstreams.

System tasks may also use standard user overlays. User overlays require no knowledge by system generation.

12.3.1 Overlay Data Structures.

The data structures used to support system overlays are shown in detail in the section on data structure pictures. The following paragraphs describe the major aspects of the structures used.

OAD - Overlay Area Descriptor

The OAD is a block of OADSZ bytes of storage immediately preceding the overlay area start address. The OAD includes information needed by the overlay loader: size of overlay, overlay number, use count, and link to the next overlay area. These pieces of data must be initialized by the subsystem planning to use the overlay. (The link and use count words are needed only for pooled overlays.) The size is the size in bytes of the area available for reading the image and relocation bit map. If the overlays are never to be relocated, the size does not have to include space for the relocation bit map. The overlay number should be initialized to -1. Immediately following the link word

OADSIZ bytes are reserved for the overlay area itself.

OVT - Overlay Disk Location Table

The OVT is a table in the system root that contains the disk locations and other pertinent information for system overlays. The copy module SOV contains indexes into this table for the information on each overlay. The table is as follows:

```
SOVT     EQU  $
E1       DATA OVTREC,OVTSIZ,OVTLOD
E2       DATA 0,0,0
   .
   .
   .
```

where:

OVTREC is the beginning record number of the overlay image on the kernel program file.

OVTSIZ is the size in bytes of the image, not including the relocation bit map.

OVTLOD is the natural load address (as assigned by the Link Editor) of the overlay.

System generation creates the OVT; IPL initializes it.

SOV - System Overlay Load Table

SOV is a template which describes the OVT built by system generation. It contains definitions for the names of all the system overlays in the system. Routines referencing overlays do so by name and copy in this module. The value of a name is an index into the OVT.

12.3.2 System Support Routines for Overlays.

The module DSC.SOVLY.SOVCPR in the system root provides three entry points for accessing overlay code. Two of these are for entering overlay code and one is for returning from overlay code.

An overlay may be called from task code using SOVLTO (link to overlay). SOVLTO preserves linkage information on the task stack. The called overlay may itself call an overlay via SOVBTO (branch to overlay). The second overlay executes, returning to the task code via the linkage preserved on the stack by SOVLTO.

SOVLTO - Link to Overlay

SOVLTO is used to enter an overlay from system task code. The caller places the overlay index from SOV in R9 and the address of an overlay area in R8. The overlay is loaded, relocated if necessary, and the code is entered in such a way that if R11 is used for a return address, return will go through SOVRFO. Hence, routines can be called that do or do not use push/pop for their entry/exit convention. Any routine that has a pointer in the entry vector can be called by this routine.

SOVBTO - Branch to Overlay
SOVBTO is used to enter an overlay from another overlay when a return path to the first overlay is not needed. It is used when continuity of logic is needed but the code will not fit within one overlay. The same R8 and R9 inputs are used as for SOVLTO.

SOVRFO - Return from Overlay
SOVRFO is used to return to the caller of SOVLTO. No input registers are needed. All registers of the returning routine, including R0, are preserved, and any error exit indicated by the contents of R0 is taken.

12.3.3 Size of Overlay Areas.

The system overlay loader does not support returning control (i.e., after calling root-resident code) to an overlay which is loaded at a different address from where it was loaded when it executed the call to the root-resident code. The system overlay loader does not support a pool of overlays.

The overlay area must be large enough for the overlay and the relocation bit map. Overlay areas can be located anywhere convenient for the subsystem. Most are allocated in the task segment for the subsystem.

12.3.4 Coding Overlays.

System overlays are coded like normal user overlays. Code is free to reference (REF) symbols defined in the overlay and in the root and to define data words referencing locations in the overlay code itself. This characteristic is achieved by relocating overlays at load time.

Each overlay must have a table at the physical beginning of the overlay that defines the locations of the routines in the overlay. When using R9 to specify which overlay to enter, the first five bits of R9 indicate which of 31 possible routines is to be used. In the following example, the following link control and modules illustrate the construction of code that is to be overlay resident. A line with three dots indicates missing code.

```
   PHASE 3,DMOV37
   INCLUDE IN.DISKMGR.OBJECT.DMOTBL
   INCLUDE IN.DISKMGR.OBJECT.DMRTN1
   INCLUDE IN.DISKMGR.OBJECT.DMRTN2
   INCLUDE IN.DISKMGR.OBJECT.DMRTN3
   PHASE ...
```

where the routines include:

```
           IDT  'DMOTBL'
           REF  DMRTN1
           REF  DMRTN2
           DATA DMRTN1
           DATA DMRTN2
           END

           IDT  'DMRTN1'     DIRECTLY CALLABLE FROM
           DEF  DMRTN1       WITHIN OVERLAY AND CALLABLE
   DMRTN1  MOV  R11,*R10+    THROUGH SOVLTO
           BL   @NFPSHX
           ...
           B    @NFPOPX
           END

           IDT  'DMRTN2'     CALLABLE ONLY BY SOVLTO
           REF  DMRTN1       CALLABLE FROM HERE BY BL
           REF  DMRTN3       CALLABLE FROM HERE BY BL
           DEF  DMRTN2
  DMRTN2   ...
           BL   @DMRTN3
           DATA X
           ...
   X       BL   @DMRTN1
           DATA Y
           ...
           B    @SOVRFO
           END

           IDT  'DMRTN3'
   DMRTN3  MOV  R11,*R10+    CALLABLE ONLY FROM WITHIN
           BL   @NFPSHX      OVERLAY VIA BL
           ...
           B    @NFPOPX
           END
```

The overlay loader is capable of handling the call and return sequence of either the standard push.pop or the enter/exit routines, as illustrated in the preceding examples.

12.3.5 Calling Routines in an Overlay.

Overlay-resident code is called with the calling sequence illustrated below. Entry from the root code into each of the routines in the preceding example is shown. All registers except R0 are transmitted to the called routine unchanged. R0 is cleared.

```
COPY DSC.TEMPLATE.ATABLE.SOV
...
LI   R9,DMOV37     TO ENTER DMRTN1
LI   R8,<overlay area address>
BL   @SOVLTO
DATA <error return>
...
LI   R9,DMOV37+>800     TO ENTER DMRTN2
LI   R8,<overlay area address>
BL   @SOVLTO
DATA <error return>
...
```

12.3.6 Internal Design Considerations.

The entry SOVLTO pushes the return address onto the current run-time stack. Calling one overlay from another is not supported in that the stack does not have information on which overlay to load and therefore the first overlay is not loaded on the return path.

When an overlay is loaded into memory, a check is made to see if relocation is needed. If so, the relocation bit map is also read in. If the load address of the overlay and the address of the overlay area in which to load it are equal, no relocation is performed. Otherwise, relocation may be needed. Each location indicated in the bit map is examined. If the reference is less than the natural load address of the overlay (computed by the Link Editor), relocation is not necessary for that location. If the reference is greater than or equal to the natural load address, the reference is altered by the following formula:

$$NRV = ORV - (NLA - ALA)$$

where:

NRV is new reference value
ORV is old reference value
NLA is natural load address
ALA is actual load address

Hence, the overlay area must be large enough to contain both the bit map and the overlay, when the overlay area is part of a pool.

12.4 WRITING AND LINKING A PASCAL SYSTEM TASK

System tasks written in Pascal may be written in several ways, depending on what portions of the task are in Pascal. If the entire task is written in Pascal and enough space is available to accommodate the Pascal run-time support, the task can make use of Pascal routines for stack and heap management, as well as other run-time support.

If task space is not abundant, Pascal run-time routines for stack and heap management can be replaced by other routines to economize on space. These routines define entry points that are replacements for labels known to the Pascal base routine, R$TASKDP. The routines are described in the paragraphs that follow.

UTR$ST

This module in DSC.UTCOMN.SOURCE.UTR$ST contains routines that perform stack and heap initialization and that perform termination processing. The routines are named R$GSHS, R$GSHP, and S$STOP. The module references labels RSTACK, STKSIZ, HEPSIZ, MDNAME, and CLNUP that are defined in a data module supplied by the user.

Parameters Module

If using UTR$ST for stack and heap management, the user must build a module that defines the parameters RSTACK, STKSIZ, HEPSIZ, MDNAME and CLNUP. The CLNUP parameter is optional; it specifies the address of an end action routine. A label MDNAME can be defined to contain the ASCII string which should be shown in a message to the system log when the task aborts. For example, the following portion of code in DSC.LOG.SOURCE.LGADAT sets the parameters for the DNOS accounting log processor. In this example DATAM is a macro that generates the number of occurrences, specified in argument two, of the value in argument one.

```
            IDT   'LGADAT'
            DSEG
            DEF   RSTACK,STKSIZ,HEPSIZ,MDNAME
    MDNAME  TEXT  'LGACCT'
    RSTACK  DATAM >3333,200         USE 200 WORDS OF STACK
    STKSIZ  EQU   $-RSTACK
    RHEAP   DATAM >3636,300         AND 300 WORDS OF HEAP
    HEPSIZ  EQU   $-RHEAP
            DATAM >4242,32          MARGIN BUFFER
            END
```

When the Pascal task is linked, the modules which replace the Pascal run-time modules must be explicitly included so that they override the default modules collected from the run-time library. The following example link control file for building a task includes its own parameters module (LGDATA) from VOLOBJ.LOG.OBJECT and the UTR$ST module from VOLOBJ.UTCOMN.OBJECT. It is the link stream for the log formatter task from DSC.LINK.SYSTEM.LGFORM.

```
NOPAGE
NOSYMT
LIBRARY    VOLOBJ.LOG.OBJECT
LIBRARY    PASCAL.MINOBJ,PASCAL.LUNOBJ,PASCAL.OBJ
LIBRARY    VOLOBJ.UTCOMN.OBJECT
LIBRARY    VOLOBJ.PASASM.OBJECT
PROCEDURE DUMROOT
DUMMY
INCLUDE    VOLOBJ.S$SGU$.DUMROOT
PHASE 0,LGFORM,PROG >C000      ; SYSTEM LOG FORMATTER TASK
INCLUDE (R$TASKDP)
INCLUDE (LGFORM)
INCLUDE (LGDATA)
INCLUDE (UTR$ST)
INCLUDE (UTPTCH)
END
```

An alternate to building the parameters module such as LGDATA is to make use of the Pascal exception handler. In this case, the text for MDNAME must be defined in an assembly language module so that messages to the system log have a module name text. The routine ONEXCEPTION must be declared in the Pascal task as:

```
   PROCEDURE ONEXCEPTION(HANDLER_LOCATION: INTEGER); EXTERNAL;
```

When the task aborts, this procedure is called, passing to it the location of the Pascal procedure which is to perform the cleanup processing. The exception handler has access to any variables in common or in the main program's stack frame, but other stack frames cannot be assumed to still be accessible. Optionally, the exception handler may be declared to accept a single integer parameter, which will then be set to the internal message number of the error condition.

The following example shows how a procedure CLEANUP might be used to handle exceptions. In this case, it merely writes a message ERROR IN PROCESSING to file F.

```
PROGRAM EXAMPLE;
VAR F: TEXT;
PROCEDURE ONEXCEPTION(HANDLER_LOCATION: INTEGER); EXTERNAL;
PROCEDURE CLEANUP;
   BEGIN
      WRITELN(F, 'ERROR IN PROCESSING');
   END;
BEGIN
   REWRITE( F);
   ONEXCEPTION( LOCATION ( CLEANUP ) ):

       main body of program

END;
```

12.5 DETAILS OF DNOS SYSTEM TASKS

The kernel program file (named according to the system name specified during system generation) and the utilities program file .S$UTIL include a number of system tasks that support execution of DNOS. Those which carry a section indication in Table 12-1 are described in detail in that section of this document; those with no section number are described only in this table.

Table 12-1 DNOS System Tasks

Task Name	Section	Purpose
DEBUG	16	Aids in debugging system code
DIOU	10	Performs device I/O utility functions
DISKMGR		Performs the Disk Management SVCs
FILEMGR	11	Performs file management operations
INV		Performs the Initialize New Volume SVC
IOBREAK	12	Performs the break key function
IOTBID	10	Bids a task from DSR
IOU	10	Performs I/O utility operations
IPC	10	Swaps IPC data for tasks that do not simultaneously fit in memory
IUV		Performs the Install Disk Volume and Unload Disk Volume SVCs
JOBMGR	8	Performs job management
LGACHN	14	Puts spooler data in the accounting log
LGACCT	14	Formats accounting log messages)
LGFORM	14	Formats system log messages
LGGLOG	14	Recovers system log data after crashes
LGRCRT	14	Creates log files
LOGON	12	Processes user log-on procedure
NAMMGR	10	Performs name management SVC
PMOVYL		Performs the Load Overlay SVC
PMPASP		Performs the Assign Program File Space SVC
PMPDEL		Performs the SVC processing for Delete Task, Delete Procedure or Program Segment, and Delete Overlay
PMPINS		Performs the SVC processing for Install Task, Install Procedure or Program Segment, and Install Overlay
PMPMAP		Performs the Map Program File SVC
PMRWTK		Performs the Read/Write Task SVC information transfer
PMSBID		Processes the Scheduled Bid Task SVC
PMSBUF		Processes the Modify BTA/JCA Size SVC
PMTBID	9	Processes Bid Task SVC
PMTERM	9	Cleans up a task that has terminated abnormally
PMTLDR	9	Loads tasks into memory
PMWRIT	7	Writes modified segments to disk
RPRCP	12	Processes Return Code Processor SVC
RESTART	12	Establishes initial system conditions
RESTART2	12	Establishes initial system conditions
SAVRES	10	Saves and restores name manager segments to and from disk

The remaining system tasks in S$UTIL support SCI and utility functions and some of these are described in detail in the DNOS SCI and Utilities Design Document. Table 12-2 lists the system tasks that support SCI or utilities.

Table 12-2 System Tasks to Support SCI and Utilities

Task Name	Function
CRV	Checks and resets disk volumes
LTS	Lists terminal status
MLP	Modifies LUNO protection
OPERATOR	Channel owner for the operator interface
RAL	Releases all LUNOs for a job
SCS	Shows channel status
SCU	Performs system configuration commands
SIS	Shows I/O status
SJSSTS	Shows job and task status
SMM	Shows system memory map
SMS	Shows memory status
XJM	Monitors execution of jobs
XPD	Displays performance data

Many other tasks found in the S$UTIL program file are not system tasks, but they do support SCI and utilities. They are described in the SCI and Utilities Design Document.

The paragraphs that follow describe some of the system tasks that are part of DNOS but are not described elsewhere in this document.

12.5.1 Log-On Task (LOGON).

The log-on task is bid by IOTBID whenever a command definition table (CDT) gives the LOGON ID as the primary task to be bid. The supplied log-on task can be replaced by a user-written task if the user wishes to have a different system environment than that provided. That task must be a system task if it is to examine system structures for currently executing jobs.

One of the special responsibilities of LOGON is to initialize the system time and date. The first time LOGON is bid, the year field (kept in the CSEG NFCLKD as YEAR) is zero. LOGON prompts for time and date and initializes the system to the data supplied. In subsequent bids of LOGON, the year field is checked; if it is nonzero, no time and date are queried.

The supplied log-on task solicits a user ID, passcode, account ID, and job name from the terminal performing the bid if the .S$SCA file indicate that log-on is required at that terminal. This data is kept for each terminal, and it is modifiable by a Modify Terminal Status (MTS) command to SCI. Before the job is started, the break key sequence is enabled at the terminal being used.

If a job already exists for the user ID, passcode, and job name given, and reconnect is specified for the terminal, the user is asked if he wishes to reconnect (that is, run in the same job). If the user answers affirmatively, LOGON bids the desired task under the already existing job.

If the user does not want to reconnect to an already existing job, or if the job named does not already exist, LOGON issues a Create Job SVC to start a job at the terminal being used. Among the parameters supplied in the SVC are the segment identifier of the name manager segment, the initial task to be bid, program file LUNO required, and any bid parameters passed from the CDT.

If the user makes an error in supplying log-on data, the Create Job SVC fails. In the case of failure, LOGON prompts again for the log-on data. A maximum of three attempts is allowed before LOGON terminates. Exceeding the limit is logged to the system log so that violations of system security may be monitored.

For some log-ons, the data gathered from the user is different. If the CDE for the key used to log-on indicates that even loading is to occur, log-on establishes the new terminal within an existing job. If the CDE indicates that a default user ID should be used, LOGON will not prompt for user ID and passcode. Using the appropriate ID and job name, the terminal is then set up within one of the jobs specified by the CDE.

The S$SCA file contains information referenced by the LOGON task to determine mode and the proper LOGON prompts for terminals in a system. An S$SCA entry is created by the Modify Terminal Status (MTS) processor. The first field in the record is the device number associated with a specific terminal. The next four fields contain the default values for the user ID, account number, passcode and/or job name if they are not prompted for at logon. Finally, an entry contains eight flags (Y for YES and N for NO) that specify the following options:

* Login required

* VDT mode

* Do not solicit job name

* Reconnect disabled

* Solicit account number

* Solicit name

* Manager files

- Terminal off

- VDT mode default

If an entry has not been created for a station through the use of MTS, the following defaults are used:

LOGIN REQUIRED	- NO
VDT MODE	- NO
DON'T SOLICIT JOB NAME	- NO
RECONNECT DISABLED	- NO
SOLICIT ACCOUNT NUMBER	- NO
SOLICIT NAME MANAGER FILES	- NO
TERMINAL OFF	- NO
VDT MODE DEFAULT	- NO

The S$SCA file is also used by the List Terminal Status (LTS) processor. If an entry does not exist, LTS outputs the defaults.

12.5.2 System Initialization Tasks (RESTART and RESTART2).

The initialization task is the first task to run after IPL. It checks for the existence of the system log files. If they do not exist, RESTART attempts to create them. If the files cannot be created, RESTART outputs a message to the system log device. RESTART then attempts to create the accounting log files. If the accounting log files cannot be initialized, a message is output to the system log.

RESTART then initializes the capabilities list file, S$CLF and restores the global logical name segment. It deletes temporary files and bids all global channel owner tasks which support DNOS and the utilities. RESTART then bids a user-defined initialization task if one was specified during system generation.

12.5.3 RPRCP.

This task processes the Return Code Processor SVC (SVC >4C). It retrieves variable text information from the call block supplied as an argument in the call. If the supplied call block has an error for which the message requires variable text and the calling task has supplied a variable text buffer, then RPRCP extracts the appropriate variable text from the call block and places it into the buffer. To do the extraction, RPRCP follows a set of tables found in the module DSC.REQPROC.SOURCE.RPRCDA.

RPRCDA includes an entry for each SVC message in DSC.MESSAGES.TEXT.SVC that has variable text as part of the message. In addition, the table has entries for the following:

- * Each SVC which can return an I/O error has an entry for each return code which is not to be replaced by the corresponding I/O message. For example, disk manager error >202A must appear in the table since the disk manager SVC can report I/O SVC errors, but 2A is a special disk manager error which is not to be replaced by the corresponding I/O error. This error has no variable text and appears in the table with such an indication. The exceptions to this rule are F3 and F4, which do not need to be duplicated for each SVC.

- * The last code in the table must be a dummy entry to allow error codes >F0 and >F3 to generate the same output for all SVCs.

If an error is encountered while processing the variable text, searching for an LDT, or accessing a task segment the error condition is returned as an error of the >4C SVC. This allows the calling task to terminate and pass back the error condition without doing error checking on the error processing SVC. If the SVC block passed to RPRCP has no error byte, a similar 4Cxx error is returned.

A number of special cases are considered by RPRCP. The Poll Task Status SVC (SVC >35) returns status (error) byte 00 when the status message needs to specify the state in byte 3. When RPRCP detects the Poll Task Status, it retrieves the state information, passes it back as variable text, and exits the RPRCP code.

Another special case is that of Activate Suspended Task (SVC >07) and Activate Time Delayed Task (SVC >0E). Each of these can return a status byte of 00 as a meaningful status message. These cases are passed along for scanning in the RPRCDA table. All other SVCs with status bytes of 00 cause RPRCP to report a 4Cxx error saying that the passed error byte is not an error.

The cases of F0 and F3 error bytes all use the same table entry, passing back the SVC code byte as variable text for the message that indicates that an unsupported SVC was issued (for the F0 error) or that a privileged SVC was issued (for the F3 error).

A number of SVCs have no room for an error byte. The table named EXCTBL is searched to see if the supplied SVC block uses one of these exceptions. If so, the 4Cxx error indicating no error byte is returned to the caller.

For all other cases, the table in RPRCDA is scanned for the specified SVC and status byte. If an entry is found, the appropriate variable text is inserted into the buffer. If no entry is found, the SVC is examined to see if it might return I/O errors. This test is made by scanning the table IOTBL in RPRPC. If the SVC code in the supplied block is in this table, variable text is built using the SVC code and status byte for the message number which indicates an I/O error in processing another SVC.

The table in RPRCDA carries a two-word entry for each SVC code, status byte combination that RPRCP must find. The first word is the SVC code, status byte pair and the second word is the address of decoding information. The decoding information is composed of the following types of entries:

1. NOVAR - No variable text is needed; this is used for those cases that must appear in the table to mask replacement by I/O error codes

2. Cxy - Convert the byte of information at offset xy in the call block into hexadecimal ASCII for variable text

3. Dxy - Convert the byte of information at offset xy in the call block into decimal ASCII for variable text

4. Eabxy - Echo the ab words of data at offset xy in the call block exactly as they are for variable text (xy must be an even offset into a call block on an even boundary)

5. P - Use the address as offset 22 into the call block as a pathname pointer and move the pathname for variable text

6. L - Use the LUNO at offset 3 into the call block to search the LDT list and return the resource name for variable text

Rules about combining these codes into legal combinations are found in the RPRCDA module.

12.5.4 IOBREAK.

The IOBREAK task performs the hard break function. It uses the following order to cause tasks to terminate: foreground down to 1 task, then background tasks, then the last foreground task, with highest priority first.

In addition, the following rules hold:

* Skip file manager

* Skip self

* If this task already being terminated by hard break, stop

* Skip task at different terminal

* Skip non-leaf mode task (that is, kill at lowest level first)

SECTION 13

SYSTEM GENERATION UTILITY

13.1 OVERVIEW

System generation (sysgen) is the process of creating an operating system. This process involves specifying all necessary features, describing the devices that will be available to the new operating system, and constructing an operating system with the stated features and devices. The difficulty of this process is compounded by the flexibility required in the sysgen software to configure only the desired features into the operating system.

The DNOS user is given a utility called SYSJEN, which asks questions that may be answered in English. These questions determine which features will be included in the new operating system and which devices will be used. The SYSJEN utility collects this data and produces a file called a configuration file. This file contains all of the information needed to construct an operating system with the desired features. SYSJEN also produces a file that describes the data structures needed to produce an operating system with the desired features in the internal format of the new operating system. This file is called the source file. The parts needed to complete the construction of the operating system with the desired features are then chosen by the SYSJEN utility. This information is placed in the link control files. These files describe which modules are needed in the operating system and the order in which they are to be used. If communications devices are to be included, these files also describe which modules are to be included in the communications device service routines and the communications software scheduler. Combining these modules in the desired way is controlled by a batch stream. After the batch stream combines the different parts of the new operating system, the system is available for use.

13.2 SYSJEN STRUCTURE

SYSJEN is a Pascal task, consisting of a root phase and three overlays. The flow of execution is from the first overlay to the second, and from the second to the third. The root contains the main driver for the program, a collection of support routines needed in more than one overlay, and all of the I/O routines.

The main driver consists of a loop that calls other routines. This main loop is called the INQUIRE loop. SYSJEN begins by loading the initialization (INIT) overlay and calling procedure INIT. The second overlay, INTERACT, is then loaded, and the main loop is entered. Another loop calls ASKQST to ask system questions. When all system questions have been asked, DEFSTR is called to define system structures, devices, jobs, channels, XOPs, and SVCs. DEFSTR is called from the main loop so that whenever a user changes a system question, he is asked that question immediately, even if he was in DEFSTR at the time of the change. The stop routine, STOPRT, is also in the main loop; this permits the user to change answers after entering the stop routine (for example, when a warning message is issued from STOPRT). If a user answers yes to the build question, STOPRT loads the third overlay, BUILD, and calls the major routine in that overlay, BILDRT. The error-handling routines are in the root phase; since the program is written in Pascal, the run-time support routines are also in the root.

The INIT overlay initializes all of the data structures used by SYSJEN, opens the JENDAT file, and opens the interactive device. This overlay also verifies that the JENDAT file is the correct one for the program. The device characteristics of the interactive device are read to make the listing routines work for that device. Many data structures are initialized in the INIT phase. The data needed to define devices is kept as sets of device types, device names, and PDT names. The locations of questions in the JENDAT file are stored in array QTX, and the type of each question is placed in array TQ. The "preanswered" questions are answered and marked as nonlistable. Next, the implication tables are filled. Pascal heap space needed for names and pathnames is allocated. The XOP and SVC tables are emptied, and all list headers are made NIL. The flow of control is linear.

The INTERACT overlay contains all routines that ask questions about system data and system structures. The command mode routines are also in this overlay, since they are interactive. The major routines ask the system parameter, device, XOP, and SVC questions. The command mode routines permit the user to change, add, and list the previously entered data. This overlay also contains the module that reads old configurations, since they appear to be interactive responses.

The INTERACT overlay has a complicated flow structure, since the command mode is called from the TRAN routine, which parses user answers; also, command mode calls the major routines in the INTERACT overlay. These routines in turn call the parser. The user can quickly exhaust all stack space because of this indirect recursion if the CMD key is struck repeatedly.

The BUILD overlay produces the files necessary to complete sysgen. One file produced is D$SOURCE, which contains the initial STA; the interrupt vectors; the special table area SSBs; the system JSB, PDTs, RPSDAT, and system JCA. SYSLINK, IOULINK, and DMLINK are the files that link the operating system. Other link control files for communications device service routines and the communication software scheduler are generated if there is any communication device. ALGSSTRM is the batch stream used to build the system. These files are built by calling a COPY module whose parameter is the record number at which to start the copy. This routine copies data from JENDAT to the appropriate file and makes any modifications necessary for tailoring the data to the current configuration. The processing is linear in this overlay.

13.3 DETAILS OF THE SYSJEN ROOT PHASE

The major root routines are SYSJEN and STOPRT. SYSJEN begins execution. After SYSJEN has called INIT to initialize all constants and pointers, the INIT1 procedure is called to read an old configuration, if needed. The value of the synonym INCON specifies which configuration and indicates that nothing need be read if the value returned is INCON. The program then enters a loop. The answer tables (QANS, QANSBUS) are scanned to find any unanswered questions. The first one found is then asked by calling ASKQST, with the number of the question stored in common variable QTA. If the user hits the CMD key at any time, s/he enters the command mode. (In command mode, a previous answer may be changed.) The loop begins each scan at the start of the question list so that questions with changed answers will be asked next. If no questions are found unanswered, DEFSTR is called to define devices, XOPs, and SVCs. When the routine returns from DEFSTR, STOPRT is called to ask whether to save the configuration only, or to build the system.

13.3.1 STOPRT.

STOPRT determines if a build is possible. If it is possible, STOPRT then asks if a build is desired. If a system does not have a disk defined, a warning message is given. The user can hit CMD and then INQUIRE to return to DEFSTR (to add a device). If a build is not possible (caused by typing STOP), the user is asked if a save is desired. If a build is chosen, the BUILD overlay is loaded and procedure BILDRT is called. If the save option is chosen, the save routine (SAVECN) from the second overlay is called and the configuration listing is produced.

13.3.2 Support Routines.

Two types of support routines are included: I/O routines and string manipulation routines. The majority are I/O routines.

The I/O routines perform the calls needed to read data from the JENDAT file, to read the input configuration, to backspace the input configuration, to write the output files, and to read and write to the user station. The JENDAT records are always read into the same buffer, JMSG. The sequential file routines always use buffer TXT, whether reading or writing. The interactive writes use buffer BUF. The interactive reads use buffer RPBUF. The sequential call blocks are allocated from the heap on opens and released on closes. The JENDAT and interactive call blocks are in common variables G and CALLBLK, respectively.

The I/O routines, their uses and parameters are as follows:

* GJ(X) - Read JENDAT record number X

* LONGPR(X) - Read JENDAT record numbers starting at X, and display to user until logical link is 0

* PRINT - Direct output to the user or batch listing file .S$SGU$.<NAME>.ERRFIL

* RSET(LUNO) - Open a sequential file for reading

* REWRIT(LUNO) - Open a sequential file for writing

* CLOS(LUNO) - Close a sequential file

* MEOF(LUNO) - Detect end-of-file

* BKSPAC(LUNO) - Backspace the LUNO

* RWSEQ(LUNO) - Read/write buffer TXT

* ENCOD(STR,LOC,X,P,B) - Write X into field STR, starting at LOC in base B, with P digits of precision.

* DECODE(STR,LOC,STAT,VALUE) - Read number VALUE from field STR, starting at the location LOC, putting the status of the operation in STAT.

The string manipulation utilities add strings to the output buffer TXT. They all use the common variable BC to point to the location at which the insertion begins in the text. The routines and their uses and parameters are as follows:

* ADDNMB(X) - Insert decimal number X (left justified)

* ADDHEX(X) - Insert X as a four-digit hexadecimal number (includes >)

* ADDNAM(X,LEN) - Insert the name X of length LEN

* ADDPAT(X) - Insert a pathname to which X points

13.4 DETAILS OF THE INITIALIZATION PHASE

There are six routines in the INIT overlay: INIT, INITCN, INITDB, INITHD, INITAL, and INITOP.

* INIT - Opens the JENDAT file and verifies compatibility between the file and the program. It also opens the interactive device and checks the values of synonyms assigned by the SCI command procedure used to start sysgen.

* INITCN - Loads the arrays of constants used by SYSJEN

* INITDB - Initializes list headers, implication tables, the XOP tables, all answers to system questions, questions unanswered by the user, questions that are nonlistable, and questions that are preanswered to be false

* INITOP - Initializes the SVC tables

* INITAL - Assigns LUNOs to input and output files

* INITHD - Initializes preanswered questions

13.4.1 INIT.

INIT first finds the value of $$MO. This indicates whether the program is running in batch mode. Then, the value of $CSNAM is found to indicate the name of the system being built. The station is opened next. A Read Device Characteristics SVC is issued to find the number of lines used by the device. The listing routine uses this number to get a maximum amount of data to the screen before waiting for the user to signal acknowledgment. Finally, JENDAT is opened, and the first record (record 0) is read to find the version number of the file. If an incompatibility is found, an error message is sent to the user and processing stops. If no problem is found, INITCN is called, followed by a call to INITDB.

13.4.2 INITAL.

INITAL creates the S$SGU$ directory on the target disk, if one does not exist. It then creates the configuration directory in the S$SGU$ directory, if one does not exist. It then creates all output files needed by sysgen and assigns LUNOs for use by the program.

13.4.3 INITCN.

INITCN begins by printing an informative message to the user indicating that execution has begun. Next, the array indicating the presence of TILINE devices (TILPRE) is set to zero. All positions on the seven expansion chassis are marked unused. Six sets of devices are filled with devices containing the desired characteristics. These are LEGALDEVICES, TILINETYPES, XCESSREQUIRED, TIMEOUTDEVICES, CHARQTYPES and ASYNCTYPES. The array of device names (DEVNAM) is filled. Array QTX is initialized with constants from file JDICONS. QTX holds the pointers to question text in the JENDAT file. TQ, the array of system question types, is the last array to be filled in this routine.

13.4.4 INITDB.

INITDB initializes all answers to system questions (QANS), questions unanswered by the user (QANSBUS), all questions that are nonlistable (QLIST), and all questions that are preanswered (QHASDEF) to be false. Next, all implications are cleared (that is, YESINF and NOINF are made false). Now the implications are made. All names and pathnames for system questions are allocated from the heap. The list headers are made null, and the array of PDT names (PDTNAM) is initialized with text.

13.4.5 INITHD.

INITHD establishes preanswered questions. QANS is made true or false (as needed), QANSBUS is made true, QHASDEF is made true, and NUMBIN is filled where needed. (NUMBIN holds either a number or a pointer to a name or pathname.)

13.4.6 INITOP.

INITOP begins by initializing all SVC information to false. The array, (DESVC), is two dimensional; of size NOSVCGROUPS by SVCPARMS. NOSVCGROUPS is a small integer constant, currently 14. SVCPARMS is a scalar, with components DESIRED, and REQOND. These components represent whether the user wants to include the SVC and whether it is required. The array of SVC IDs (OPSVC) is initialized to false. This array is used to build RPSDAT and to decide which optional processors to include in the link stream built in the BUILD phase. INITOP then initializes DESVC by assigning those values that are initially true. Next the SVCGRP array is filled. This array is used in the BUILD phase to decide which SVCs have been chosen. A value of 0 indicates that the SVC is required, -2 indicates nonexistence, -3 indicates that the SVC is included if the common option is chosen, and other numbers indicate an SVC group chosen by group name.

13.5 DETAILS OF THE INTERACTIVE PHASE

Major routines in the INTERACTIVE phase are ASKQST, DEFSTR, COMMND, CHANRT, DELRT, and LISTRT. The translation routine, TRAN, is also in this overlay and is called by any routine needing a response parsed. Several other support routines are available throughout the phase.

13.5.1 General Support Routines.

TRAN

TRAN parses the user response and returns a token in common variable TR. If the user terminates a call with the CMD key, TRAN directly calls COMMND. This routine removes blanks from the answer. It also intercepts a STOP any time one is entered by a user.

COMMND

COMMND calls TRAN on each user command entered. Then the proper routine is called to perform the desired function. The routines called from COMMND are CHANRT, DELRT, and LISTRT. The INQUIRE command causes an escape from the COMMND routine.

SETUP

SETUP is used by INIT1 to convert answers read from a configuration file to what looks like interactively collected data.

Error Routines

Two error routines in the INTERACTIVE phase are ERRMSG and PUNT. ERRMSG produces the set of questions and error messages that result from answering a question incorrectly that was asked by ASKNAM, ASKNMB, ASKYN, OR ASKPAT. If the error occurs while a configuration is being read, PUNT is called to indicate which record was in error. PUNT terminates the program.

13.5.2 Asking System Questions.

The ASKQST routine asks all system questions by looking for the question type of QTA in array TQ. Then, one of the five prompting routines (ASKNAM, ASKNMB, ASKYN, ASKPAT, ASKELM) is called to prompt for and validate the answer. This is one of two major drivers in the INQUIRE mode. These five routines all use the numeric fields of the JENDAT file for information about acceptable answers to the question. The fields that contain answer information are DEF, AAT, LB, UB and NEXT. DEF contains a default answer. AAT is the acceptable answer type. LB and UB are lower and upper bounds on the answers. NEXT is the record number which logically follows this question.

ASKNAM

ASKNAM is used to ask name questions. This routine uses the DEF field to represent the ordinal of the default answer, if any exists. The possible values for AAT are as follows: 0 indicates that any answer is acceptable; 1 indicates that the only acceptable answer is either LB or UB (that is, an answer must begin with a letter whose ordinal is LB or UB); and 2 indicates that the question must be answered. If DEF is 0, no default answer exists.

ASKNMB

ASKNMB uses DEF as the default numeric answer. The possible values for AAT are as follows: 0 indicates that any answer is acceptable; 1 means that an answer must be larger than LB to be acceptable; 2 indicates that the answer must be between LB and UB inclusively; and 3 indicates that the only acceptable answers are LB or UB. This routine is a function.

ASKYN

ASKYN uses DEF in a type transfer to Boolean values as the default answer, such that 0 represents false and 1 represents true. The possible values for AAT are as follows: 0 indicates that a default exists and 1 indicates that no default exists. This routine is a function.

ASKPAT

ASKPAT uses the logic of ASKNAM for producing prompts and error messages. Then, this routine checks for valid pathname syntax, producing any prompting necessary because of pathname syntax.

ASKELM

ASKELM uses the logic of FNDANS. It will not stop unless the answer is found or the command key is entered.

FNDANS

FNDANS starts with record number NEXT, comparing the TXT field of each record with the answer supplied by the user. When a match is found, the DEF field is returned as the answer. FNDANS is a function, returning false if no match is found.

13.5.3 Defining Structures.

DEFSTR, the other driver in the INQUIRE mode, defines devices, XOPs, and system-supplied optional supervisor calls. The routines used are ADDDEV, ADDXOP, and ADDSVC. They include the logic needed to call the prompting routines for user interaction, answer validation, and error message production.

ADDDEV

ADDDEV asks questions that allow the user to define devices. Also included are the routines ADV2, ADV3, ADV4, DEVINT, RENAME, ADDTPD, ADDVDT, ADDSD, and ADDCOM, which are continuations of ADDDEV. ADDDEV initializes a local copy of an empty device definition and calls DEVINT to ask interrupt and address-related questions. ADV2, ADV3 and ADV4 are called to ask other device questions. These routines ask all of the unanswered questions. Then, RENAME is called to revison bar off name this device. If the user answers all questions, a permanent copy is created from the heap and linked to all other devices on a singly linked list. ADDTPD is called to ask more TPD questions. ADDVDT is called to ask more VDT questions. ADDCOM is called to ask more communications device questions. ADDSD is called from ADV3 to ask more special device questions.

The declaration for a device is as follows:

```
DEVICE = RECORD
    LINK              : DEVPTR;
    DVTP              : DEVTYPE;
    INTRPT            : INTEGER;
    POSITION          : INTEGER;
    CHASSIS           : INTEGER;
    SYSNAME           : ARRAY [1..4] OF WRD;
    TIMEOUT           : INTEGER;
    CHARQ             : INTEGER;
    XCESS             : BOOLEAN;    (* RECORD = TRUE *)
    CRUBIT            : INTEGER;
    NODR              : INTEGER;
    INTERFACE         : INTEGER;
    CHNNMB            : INTEGER;
    ADDRESS           : INTEGER;
    CASE DEVTYPE OF
    DS, DK  : ( RECSIZ     : INTEGER
              );
    LP   : ( WIDTH      : INTEGER;
             PMODE      : BOOLEAN;  (* SERIAL = TRUE *)
             XCHAR      : BOOLEAN;  (* EXTENDED = TRUE *)
             MULATOR    : INTEGER;
             LP_SPEED   : INTEGER;
         );
```

```
  VDT : ( VDT_TYPE                 : INTEGER;
          GOT_A_PRINTER            : BOOLEAN;
          SPEED                    : INTEGER;
          SWITCHED                 : BOOLEAN;
          OUTPUT_FIFO              : INTEGER;
        );
  KSR : ( TERMINAL_TYPE   : INTEGER;
          BAUD_RATE       : INTEGER;
          ACU_PRESENT     : BOOLEAN;
          ACU_ADDRESS     : INTEGER;
          ECHO            : BOOLEAN;
          FULL_DUPLEX     : BOOLEAN;
          COMM_INTERFACE  : INTEGER;
          SWITCHED_LINE   : BOOLEAN
        );
  ASR : ( CASXCESS    : BOOLEAN     " RECORD = TRUE
        );
  COM : (
          BOARDTP     : COMBRDTP;
          USRBRDTP    : COMBRDTP;
          SPNAME      : WRD;
          BUFSIZ      : INTEGER;
          NOMDL       : INTEGER;
          IPCNOSES    : INTEGER;
          PROTOCLS    : ARRAY [0..3] OF PROTO_REC;
        );
  SD  : ( SDNUMB      : INTEGER;    " SD NUMBER
          SDTIL       : BOOLEAN);   " TILINE DEVICE
  VT  : ( VTNUMB      : INTEGER);   " # OF VT

END;
```

ADDSVC

ADDSVC adds system optional SVCs to the user-generated system. This routine asks for a group name. The user enters an abbreviation or the whole name, and ADDSVC calls FNDANS to find the group number of that group name. If this group is available on the generated system, it is added to the table of desired SVCs (DESVC). Otherwise, an error message results. The definition of the DESVC is as follows:

```
SVCPARMS = ( DESIRED, REQOND, NSONC);
DESVC : ARRAY [1..NOSVCGROUPS,SVCPARMS] OF BOOLEAN;
```

ADDXOP

ADDXOP asks for the level of the XOP first. If that level is available the user is asked for the entry point of the XOP processor, the workspace pointer (WP) of the processor, and the pathname of the object code of the processor. The information is kept in the table XOPA. The pathname is kept in the heap as are all other pathnames. The definition of XOPA is as follows:

```
XOPVECTOR = RECORD
     HERE     : BOOLEAN; " XOP DEFINED
     XOPPC    : ASMNAM;  " ENTRY POINT LABEL
     XOPWP    : ASMNAM;  " WORKSPACE LABEL
     XOPNAME  : PPTH     " PATHNAME OF OBJECT POINTER
     END;
XOPA : ARRAY [0..14] OF XOPVECTOR;
```

13.5.4 Changing Structures.

To change a structure, the user must first identify it. The system then searches for the specified structure. If it is found, it is deleted and redefined. The routine CHADEV handles this processing for devices, while CHANRT handles it for XOPs. CHANRT calls routines FNDQST, CSYQST, FNDXOP, and ADDXOP. CHADEV calls FNDDEV, DELDEV, RENAME, and ADDDEV.

FNDQST

FNDQST is used by CHANRT to decide which question is being abbreviated by the user. The questions are scanned sequentially until one that has been correctly abbreviated is found. To correctly abbreviate a keyword, both the keyword and the abbreviation must begin with the same letter. All letters in the abbreviation must appear in the keyword and in the same order. The first letter following a blank in the keyword must match the first otherwise unmatched letter in the abbreviation. If the abbreviation is unmatched, the value returned by this function is false.

CSYQST

CSYQST changes system questions. If the question is currently preanswered, it is marked unanswered and made listable. Any other questions between this question and the expansion card questions are unanswered.

FNDDEV

FNDDEV asks for the device name and searches the device list for a device with that name. The pointer to the device is returned, if found, or an error message is produced.

FNDXOP

FNDXOP verifies that a given XOP has been defined on the level that the user enters. If that level has not been defined, an error message is produced. If the XOP was defined, the value is returned in common variable CXOP.

DELDEV

This delete routine deletes an item by linking around the item and disposing of the heap space used by the structure. Devices are kept in singly linked lists in the heap. XOPS are kept in an array in common; consequently, they require no delete routine.

13.5.5 Deleting Structures.

DELRT operates similarly to CHANRT except that DELRT never issues a call to add a new structure; it issues calls only to the routine that finds the desired item and to the routine that deletes that item.

13.5.6 Listing Structures.

LISTRT is the routine that produces configuration listings. These listings may be produced at the station for viewing by the user or to a file to be read by another use of SYSJEN or for later reference. The routines used by LISTRT are FORM, SHOWDV, and LIST2. LIST2 is a continuation of LISTRT, which calls FRMXOP. These routines all call DUMYUP and LOOK which transfer text from the JENDAT file to the output buffer, eliminating string constants from the program. All of these routines call ADDHEX, ADDNAM, ADDNMB, and ADDPAT for string operations. DISPAT is called by all of these routines. It directs the output to either a file or the interactive device. If the program is in the INTERACTIVE phase, the user sees the list; otherwise, the information is written to the CONFIG file.

FORM

FORM is used to fill in the answers to system questions. The type of question is found in array TQ, and the answer is added to a line of text read from the JENDAT file.

LOOK

LOOK reads a record from JENDAT, scans for the question mark in the text field, and initializes common variable BC to point to the question mark. Thus, answers may be encoded into the field.

SHOWDV

SHOWDV uses the logic of FORMDV to show all pertinent information about a device.

FORMDV, FRMXOP

FORMDV, and FRMXOP build the lines of text for displaying information about devices and XOPs to the configuration file or the user.

FILLTB

FILLTB is called after every system question is answered to update the question-answered table by implication. This routine uses common arrays YESINF and NOINF to answer certain questions for the user.

13.6 DETAILS OF THE BUILDING PHASE

The flow through the BUILD phase is linear. The first module called is INITBL, which initializes data structures that cannot be initialized as long as the user can change his mind. The next routine called is SYSJCA, which defines the system job. BLDWSR is called to build interrupt decoder tables. PDTBLD builds the PDTs. BLDSSB builds the SSBs. If communications devices are genned, BLDSWS and BLDDSR build the files required to build the communications software scheduler and communications device service routines, respectively.

Throughout the process, all of these routines and BILDRT itself depend heavily on the COPY routine. Although the COPY routine is physically split into 17 different routines, they all function as one routine. COPY is used to access data in the JENDAT file, process it as specified in that file, and output it to the files for the built system.

INITBL

INITBL marks the SVCs required by system common and system disk in addition to those that were chosen by the user. The OPSVC array is then filled by assigning the values indicated by SVCGRP and the DESVC array. The use of each interrupt level is then determined (INTUSE) and expansion chassis .use is marked in array CHSINUSE. INITBL marks the combinations of communications boards and protocols which have been chosen by the user. This information will be saved in the COMSTATUS array. If communications devices have been genned, INITBL will call INITCM to create all files required for communications link. Finally, INITBL will set the flags used o load the DSR overlays.

SYSJCA

SYSJCA scans the job list, looking for a job with an ID of 0. If such a job exists, it becomes the system job. If this job does not exist, it is created. This routine is used to prevent special case coding for the system job in later routines.

BLDWSR

BLDWSR first builds the workspaces for the interrupt decoder. This routine uses array INTUSE, which was initialized in INITBL to decide what devices are available at each interrupt level. The workspace is different for single device per interrupt, multiple device per interrupt, one or more asynchronous TILINE controllers per interrupt, and each of the expansion chassis. If an interrupt level has more than one device, a multiple interrupt decoder table is built for that interrupt level. The table consists of interrupt bit positions used for polling in the interrupt decoder, and interrupt vector pointers. The table is of variable length. If an interrupt level has one or more asynchronous TILINE controllers, a multiple interface table is built. This table consists of the controller address and a pointer to the table of channel entries. The table of channel entries consists of the interrupt vector pointers and the controller channel number. If a device has been defined on an expansion chassis, a table of interrupt vector pointers for that chassis is built. The size of the table is 24 entries. Then, all interrupt vectors are built, one for each device that shared an interrupt level on the main chassis and each device on an expansion chassis.

PDTBLD

PDTBLD is the main driver used in building PDTs. Note, however, that PDTFIL builds the fields needed to fill in the template kept in the JENDAT file, and PDTGO calls for the correct pieces of the template that are kept in the JENDAT file. PDTBLD sets either a flag in the common variable DCODE to indicate that the current device is the first device of a multiple drive controller, or a flag in common variable CASFLAG, to indicate that the current device is a cassette drive. PDTBLD also chooses the correct name to be used in filling the PDT template. PDTBLD is also responsible for producing the trailer equate for the end of the PDT chain. Note that this trailer equate is not produced if RTS is present. PDTFIL defines the values of flag words, labels, and special constants used in filling the template. These include the device type flags, device status flags, the address, and the interrupt level. PDTGO decides where to begin copying in the JENDAT file. Some PDTs have no extensions, while others have as many as three.

BLDSSB

BLDSSB builds the SGB in the system table area, then builds SSBs for the segment management table area, file management table area, system root, and system common. The fields that must be initialized are the link field, run-time ID, segment attributes, use count, segment group block pointer, segment beet address, and length of the segment. This routine initializes these fields and calls the correct copy routine to build the SSBs of the STA.

13.7 JENDAT FILE

SYSJEN maintains a large amount of data in the JENDAT file, which is a random file of Pascal records. The definition of the record type is as follows:

```
  TYPE GENMSG = RECORD
                DEF  : INTEGER;
                AAT  : INTEGER;
                LB   : INTEGER;
                UB   : INTEGER;
                NEXT : INTEGER;
                LEN  : INTEGER;
                TXT  : PACKED ARRAY [1..76] OF CHAR END;
```

The uses of each field are described first for the INTERACTIVE phase of SYSJEN and then for the BUILD phase. (The uses differ greatly.) Since the field names are based on their original uses in the INTERACTIVE phase, they do not accurately portray their uses in the BUILD phase. The names were chosen from the following uses of each field:

Name	Use
DEF	Default answer
AAT	Acceptable answer type
LB	Lower bound on answers
UB	Upper bound on answers
NEXT	Record number that logically follows
LEN	Length of the text that follows
TXT	Message text

13.7.1 Interactive Use of the JENDAT File.

The length field is used in the call block as the number of characters to be written. The text field contains the message to be written to the user's screen. Five types of questions asked by sysgen determine how the fields of the record are to be used. The types of questions asked are as follows:

- Number questions

- Name questions

- Element questions

- Pathname questions

* YES/NO questions

13.7.1.1 Number Questions.

Number questions use the default field to find the answer to be used if the user hits the RETURN key in response to a question.63No conversion is necessary. The value in the AAT field signifies the following:

0 = any numerical answer is acceptable
1 = any number greater than LB is acceptable
2 = any number X such that LB <= X <= UB is acceptable
3 = only LB or UB is acceptable

This scheme allows the ASKNMB routine to determine the validity of answers without special case code for each question that requires a number answer. If only one of three or more noncontiguous numbers are acceptable, special case code is still required.

Consider the following example:

```
REC #  DEF AAT  LB  UB NEXT LEN TXT
   20   60   3  50  60   21  20 LINE FREQUENCY? (60)
```

Record number 20 asks for the frequency of the line voltage. The default answer is 60. The default is always included in parentheses in the text to inform the user of the default. Since AAT = 3, the only acceptable answers are 50 or 60. The length of the text string to be written is 20 characters. This permits the response to be accepted directly after the question, without scanning the text string. The NEXT field indicates that if the user responds by entering a ? or incorrectly answers the question, the longer form of the question is at location 21. This permits many changes to be made to the JENDAT file with no changes in program logic, and recompiling is not necessary.

Often, several records are logically followed by the same record. Such is the case when, in answering the second question, the user incorrectly defines the interrupt levels. All of these error messages may be provided with the same text, as shown in the following example. (Note that all of the text will not fit across the page in this example.)

```
REC # DEF AAT  LB  UB NEXT LEN TXT
  179   9   2   3  15  185  49 WHAT IS THE INTERRUPT LEVEL
REC # DEF AAT  LB  UB NEXT LEN TXT
  180   7   2   3  15  185  46 WHAT IS THE INTERRUPT LEVEL
```

This example asks for the interrupt level of a tape unit and then the interrupt level of a flexible diskette. The default is either 9 or 7. The acceptable answers are the same, but the question lengths differ slightly. Notice that the next logical record is the same for each question. Seven other questions also use the same text at record 185.

Often, the value of NEXT is 0. This indicates that the current record is the last of a chain. This will be the last record that the current request will display.

13.7.1.2 Name Questions.

Name questions assume that only the first character of the answer is significant. The default contains the ordinal of the first character of the name. The value of AAT is as follows: 0 indicates that any answer is acceptable, 1 indicates that only ORD(LB) or ORD(UB) is acceptable, and 2 indicates that the user must answer the question.

If AAT is 1 and DEF is 0, no default exists. As a result, the question (or a similar question) is asked until it is answered correctly. The following example asks for the type of line printer being defined. The user is expected to answer either serial or parallel. The default is serial. The ordinal of S is 83, and the ordinal of P is 70. Thus the user may hit the RETURN key to indicate serial (the default). Any answer except a RETURN or words that begin in S or P are treated as incorrect answers.

REC #	DEF	AAT	LB	UB	NEXT	LEN	TXT
272	83	1	70	83	273	20	PRINT MODE? (SERIAL)

In the next example, the user is asked for the workspace label of a special DSR. Any answer is acceptable, but note that an answer must be given.

REC #	DEF	AAT	LB	UB	NEXT	LEN	TXT
344	0	2	0	0	345	14	DSR WORKSPACE?

13.7.1.3 Element Questions.

Element questions use the default answer if AAT is 1 and the user hits the RETURN key in response to the answer. Otherwise the value of NEXT points to a list of valid answers.

The list can then be searched using ASKELM, which compares the user response to the TXT (message TEXT). If a match is found, the DEF field is returned as the answer. Otherwise, the next logical record is searched. (The next logical record is pointed to by the field NEXT. If NEXT is 0, this signifies the end of the list). This process is continued until a match is found or the CMD key is pressed. If an invalid answer is entered, then the longer form of question is asked.

An example of an element question is one that asks baud rate, as follows:

```
REC #   DEF ATT   LB   UB NEXT LEN TXT
 1820     0   0    0    0  1822 10 BAUD RATE?
```

The next physical record of the file has the first line of the explanation of the question and records starting at 1822 have the valid options. The entries for the valid responses include as the DEF field the internal value needed by SYSJEN.

NOTE

> The first line of the longer question must be physically after the short question. The valid answers must be logically after the short form. This applies only for the element questions.

WARNING

> The internal value (the DEF field) usually plays a significant role during the build (and sometimes the inquiry phase) of Sysgen. When adding or deleting elements from these lists, great caution should be used.

13.7.1.4 Pathname Questions.

Some questions are answered by pathnames. These questions use the logic of the name questions. When a valid name has been entered, (almost anything is an acceptable name), the syntax of pathnames is checked. Thus, no special entries are made in the file. In the following example, the user is forced to enter a name. The syntax requirements are treated in no special way in the JENDAT file; only the text gives any indication of the requirements of the answer.

```
REC#   DEF  AAT   LB   UB  NEXT  LEN TXT
 870    0    2     0    0  867   25 APPLICATION PROGRAM FILE?
```

In the following example (defining the KSB for a special device), the user is not forced to answer the question; this means that the device does not have a keyboard.

```
REC#   DEF  AAT   LB   UB  NEXT  LEN TXT
 337    0    0     0    0  338   11 KSB? (NONE)
```

13.7.1.5 Yes/No Questions.

Yes/No questions have as default answers the internal Pascal representation of true and false, 1 and 0. The value of AAT is as follows: 0 indicates that a default exists, 1 indicates that no default exists and the user must answer.

In the following example, the user is defining a line printer. The question determines whether an extended character set is available on that printer.

```
REC #   DEF AAT   LB   UB NEXT LEN TXT
  277     0   0    0    0  278  14 EXTENDED? (NO)
```

The default is NO, and the user may press RETURN to get the default.

13.7.2 BUILD Use of the JENDAT File.

The JENDAT file is predominantly used for building source code in the various modules that describe the system being generated by SYSJEN in the BUILD phase. The DEF field is used as follows: if DEF is 0, no processing is required on the field; if DEF is 1, AAT contains the type of modification that is required. This value is used as the case statement variable of the COPY routines in SYSGEN. The use of the values of NEXT and LEN in the BUILD phase is similar to their use in the INTERACTIVE phase. The remaining fields are of variable usage. The LB field is almost always a pointer into the record. This value is the column number of the first column that requires modification. In the example that follows, a label is inserted on a PDT.

```
REC #   DEF AAT   LB   UB NEXT LEN TXT
  378     1   0    3    0  379  13 PS     EQU  $
```

The DEF flag signals that the record must be modified. Case 0 is used to indicate adding the current device name into location LB. If the current device is DS01, this name is inserted at column 3 and the record is modified as follows:

```
  PSDS01 EQU  $
```

Generally, the UB field is 0. However, it is sometimes used to mark a second column for modifications, as in the following example where an interrupt vector is being defined.

```
REC #   DEF AAT   LB   UB NEXT LEN TXT
  594     1  29   15   29    0  57 IV    DATA      ,SG3BGN,MP
```

The device name is added at column 15, and an offset determined by the device type is added at location 30. Internal logic also adds the name at column 3 and a field determined by the device type at column 13. If the device type were VDT and the current name were ST16, the record would be modified as follows:

```
IVST16 DATA KBST16,SG3BGN,MPST16
```

Sometimes the UB field does not hold a location in a specific record; instead, it holds the current location in a section of records. For example, consider building the expansion chassis interrupt decoder table. This table is 24 records long, each containing either 0 or a label of an interrupt vector (as in the previous example).

```
REC #   DEF AAT  LB  UB NEXT LEN TXT
  830     1  28  15   6  831  14          DATA IV
```

In this example the UB signifies that the user is building the seventh entry, the entry for interrupt position 6. If the current device name was ST16, the seventh entry in the interrupt table would become:

```
DATA IVST16
```

This use of UB is prevalent in the table building portions of sysgen, although its use as a column number is more common. The only accurate means of determining its use is to examine the source in the COPY modules.

13.7.3 Sample Copy Module.

The following copy module indicates the BUILD usage from the SYSJEN program. The entries are divided into groups of ten to simplify the requirements on the compiler. The COPY module has a case statement that uses the AAT field.divided by ten to determine which sub-COPY module to call to process the record. The ATT field is a hexadecimal number. To determine which sub-COPY module is going to be used, convert ATT to decimal and divide by ten. The example given determines whether to include the given record in the link stream. The record in question should be included if the system is a DNOS/D.

```
REC #   DEF AAT  LB  UB NEXT LEN TXT
 1621     1  5F   0   0 1622  20 PHASE        3,CFDFOV
```

The AAT field of 5F indicates that case number 95 of the COPY module is called to process this record. The code is taken from COPY9.

```
BEGIN
  DISK := QANS[DISKQST];
  KIF := QANS[KIFQST];
  CD := QANS[CDFQST];
  FM := QANS[FMQST];

  CASE JMSG.AAT MOD 10 OF
    0 : PRFLAG := QANS[EXFQST];
    1 : PRFLAG := NOT FM;
    2 : PRFLAG := NOT CD;
    3 : PRFLAG := FM;
    4 : PRFLAG := CD;
    5 : PRFLAG := DISK;
    6 : PRFLAG := NOT DISK;
    7 : PRFLAG := NOT KIF;
    8 : PRFLAG := KIF;
    9 : PRFLAG := NOT QANS[EXFQST];
  OTHERWISE ; END;
END;
```

The variable PRFLAG indicates that the record should be printed if the value is true; otherwise it should not be printed.

13.8 JENDAT EDITOR

A special editing program is available to the DNOS development staff to maintain and modify the JENDAT file. This editing program is documented in the section on DNOS tools.

SECTION 14

LOGGING AND ACCOUNTING

14.1 LOGGING AND ACCOUNTING FUNCTIONS

The DNOS logging system logs information to the system log files (.S$LOG1 and .S$LOG2) and an optionally specified log device. The log is used to inform the operator or user of the state of the hardware and/or operating system. The log contains the following types of messages:

* Device errors with the controller images before and after an operation

* Device errors with the offending call block

* Abnormal task termination

* Statistics from a DSR

* Operating system informative messages

* User messages (from SVC >21)

* Cache memory errors

* Memory parity errors

The DNOS accounting system logs information concerning use of system resources by user tasks to the system accounting files (.S$ACT1 and .S$ACT2). Entries are made for the following events:

* Job initialization

* Task termination

* Job termination

* Spooler device use

* Initial program load (IPL)

* User-defined (from SVC >47)

14.2 LOGGING AND ACCOUNTING TASKS

The logging and accounting systems each have a queue server task to write data to the disk files. Whenever data is to be written to either the log or accounting files, an entry containing the data is put on the appropriate queue to be processed by the queue server. Since the function of writing data to the disk files is similar in both queue servers, several subroutines are shared between the tasks. If errors are encountered while writing to the files, then a message is generated for the log device and the attention device.

14.2.1 LGFORM.

The system log formatter task (LGFORM) serves the system log queue. This queue has system log blocks (SLBs) generated by user tasks using SVC >21 and from a variety of operating system sources.

LGFORM first ensures that logging is functional. It then dequeues one entry after another until the queue is empty. For each log entry, LGFORM builds one or more lines of system log text and outputs it to the system log file (one of two possible system files) and, optionally, to a system log device.

Since the log queue is a finite limit, there may be times when more messages are sent to the queue than will fit. In that case, DNOS increments a lost message count in the newest entry on the log queue. When LGFORM processes a log entry, it checks the lost message count. If it is greater than zero, LGFORM outputs a message showing how many log entries were lost.

Other error conditions are also monitored and reported. If the files and/or log device should become inoperative, a message is sent to the attention device specified during system generation. When one of the log files becomes full, a message is also sent to the attention device. The alternate file will then become the log file in use.

14.2.2 LGACCT.

LGACCT is the task that processes the accounting queue. For each entry on the accounting queue, LGACCT builds an accounting record and writes it to the accounting file that is currently in use. Like the log files, when one accounting file is filled, LGACCT switches to the other accounting file. Errors in the accounting files are also reported to the log attention device. Unlike the log files, the accounting files contain binary data instead of text. For information on processing the data in the accounting files see the DNOS System Programmer's Guide.

14.3 SUPPORT ROUTINES

Since many of the functions of the log and accounting system are similar, many of the following routines are linked into both systems.

LGALUN
LGALUN is used to assign a LUNO to the output file currently in use. It may be used for either the log files or the accounting files.

LGATTN
LGATTN is used to write a message to the attention device if one has been specified.

LGCHEK
If an error is encountered while writing to the log files and log device, then LGCHEK will delay and continue retrying the request until it succeeds.

LGSWTH
When an output file fills up, LGSWTH is called to close the current file and open the other file. The new file is marked as the current file. If there is a task specified to process the files, then that task is bid after the files are switched. If both a system-supplied task and a user-defined task are supplied, the system-supplied task is bid first. This routine can be used for either the log or accounting files.

LGWACC
LGWACC outputs an accounting record to the current accounting file.

LGWDEV
LGWDEV writes a message to a device. It may be used for the log device or the attention device.

LGWLOG
LGWLOG outputs a log message to the current log file. It also ensures that the log files are switched when one fills up and that the message goes to the log device if one has been specified.

14.4 MISCELLANEOUS MODULES

Several modules are used throughout DNOS to generate log entries.

LGACHN
LGACHN is a task that runs in the spooler job. It receives IPC messages from the spooler system containing device

accounting records. LGACHN is responsible for putting those accounting records on the accounting queue.

LGDEV

LGDEV is a routine in the DNOS kernel that is used to generate log messages for device errors. It calls LGQBLK after setting up a device message.

LGGLOG

LGGLOG is a task that is invoked during system restart to read the crash dump from the previous crash and retrieve any log messages that were generated before the crash but did not make it to the log file. Those messages are then written to the log file.

LGQACC

LGQACC is a kernel routine that queues an accounting record to the accounting queue.

LGQBLK

LGQBLK is a kernel routine that queues a log message to the log queue.

LGRCRT

LGRCRT is a task that is invoked when the log or accounting files need to be created or recreated. When recreating a pair of files, it ensures that the new files are created before destroying the old files.

LGSVC

LGSVC is the module in the DNOS kernel that processes user log message SVCs (SVC >21). It generates a log record to be queued to the log queue.

PMACCT

PMACCT is the kernel routine that processes the accounting SVCs (SVCs >47 and >49). For the log accounting entry SVC (>47), it creates an accounting record and queues it to the accounting queue. For the get accounting information SVC (>49), it returns the accounting information for the requested task in the call block.

SECTION 15

DNOS PERFORMANCE PACKAGE

15.1 OVERVIEW

The DNOS Performance Package uses the 990/12 writable control store (WCS) to enhance DNOS speed. This section describes the WCS component and also describes the way DNOS operates without the WCS component.

15.2 DNOS SOURCE CONVENTIONS

DNOS source has two components that enable use of WCS for speed enhancement. These are a set of XOPs and a routine in the NUCLEUS source directory named NFMAT. For a number of system routines, the execution takes place in WCS when the performance package is available and otherwise, takes place in memory (referred to as default code). The access to the system routines is via XOP instructions, placed into the DNOS source code by macros. These macros are described briefly in the section on naming and coding conventions. In addition, NFTBID ensures that all system tasks that are bid have their WCS status bits turned on. This will cause routine calls to go to microcode.

The XOP assignments are as follows.

XOP	10,R1	RPPRCK
XOP	10,R2	RPSGCK
XOP	10,R3	NFRTA
XOP	10,R4	NFGTA
XOP	10,R5	NFGTA0
XOP	11,RX	NFPOPX
XOP	12,RX	NFPSHX

It is important to note that the microcode assumes that bit 11 of the instruction is a zero. This is used to quickly decide what map file the caller is in. This means that only direct register addressing can be used in the XOP instructions. Any violation of this rule generates unpredictable results.

NFMAT is a data area that starts at location >E0. NFMAT is used to tell the microcode the location of things it wants to access in memory. For example, it contains the address of JCASTR. Since the microcode uses its own equates, the order of the data in NFMAT must not be changed without changing both the default and performance microcode modules. Data used to crash and provide addresses for the default microcode starts at location >100. Since the microcode can only map an eight bit constant, this area is addressed by shifting an eight bit number to the left one bit and using that as the address. For this reason, some addresses have equates in the microcode that are only half of their expected value. This also means that if new data is added after >100, the corresponding equate in the microcode must be divided by two to force it into eight bits and multiplied by two to generate the correct value. The end of NFMAT contains a set of ASSUME statements. These ASSUME macros are needed because the microcode will reference fields within a block (like TSBML1). If these assumes should fail, then the equates in the default and performance code must be changed.

If the microcode should ever need to crash, the PC will be placed into R11 and the crash code written to location >104. The program will then return out of microcode at location >100 which contains a BLWP to NFCRSH.

15.3 MICROCODE CHARACTERISTICS

The microcode is divided into two modules: the performance code and the default code. The performance code is used to speed up DNOS and the default code provides an interface when performance code is not installed so that a 990/12 can execute XOPs efficiently. XOPs are decoded in the following manner:

1. The XOP instruction is vectored into WCS

2. The XOP level provides a first level of decoding

3. The correct routine is found from a branch table

The default control store is used so that XOPs can be executed on a 990/12 quickly. Since it is always loaded, it is linked in with the IPL procedure. If SLWCS finds no WCS image file on disk, it moves the default code into the WCS segment. Located before the WCS object is a module called PFWCSO. This module simulates the overhead found in an image file. It contains the number of microwords in the file, the start address, and other information. If default code is being changed, PFWCSO must be changed to reflect the new length. Equates for new symbols must also be added to the microcode.

15.4 MICROCODE CODING CONVENTIONS

This section describes the standard conventions used to write microcode. It specifies the syntax, labels, and comments that microcode should contain. It is assumed that the reader has read the 990/12 Microcode Development System Programmer's Guide or that he knows 990/12 microcode language.

15.4.1 Standard Syntax For Microcode States.

Since the microassembler does not specify an order in which microcode mnemonics are written, an informal standard has been adopted for DNOS microcode. This order makes the microcode states more orderly and facilitates reading. The format of a state is:

```
     LABEL: READ,
            ALUI. XAC<A+B,
            A<ABUS ABUS1<WK R(2),
            B<BBUS BBUS<CBUS,
            CBUS<MDI,
            ABUS2<SUM AREG<ABUS,
            MAP<MC MC<MC+2,
            MDI<BTL,
            IF. ALU-EQ TRUE JUMP GTA100
```

The label is put at the top of the state. After the label comes the memory I/O mnemonic. On the next line, the ALU form is specified followed by the ALU destination and the operands. Next, the A operand is traced from the ALU to its source (in this case WK R2). The B operand is then traced from the ALU to its source (the MDI). The ABUS phase two source and destination are then given (the SUM BUS and AREG respectively). After that, the destination of a memory read is given (the MDI in this case). Following that line, any conditional or unconditional jumps are specified (IF.).

15.4.2 Labeling Conventions.

Whenever possible, labels in the microcode correspond to the labels in the assembly language version of the module. If the microcode being written does not have equivalent assembly code (like for interrupt return code), then descriptive labels must be used.

15.4.3 Commenting Conventions.

Comments have two parts. The first part of a comment is the assembly language code being emulated. For example, when emulating an AI R4,STALNK instruction, AI R4,STALNK should be in the comments. The second part is just the normal comment put in to clarify what is happening. It is also advisable that the programmer should stop periodically in the code and write a paragraph that describes what data is in what registers. Between the labeling conventions and the commenting conventions, it should be relatively easy for a programmer to read microcode when he has the corresponding assembly language available.

15.4.4 Common Routines.

There are a number of routines provided in the microcode to perform common functions. The programmer may use them or, if faster, execute them inside his own states. The routines are:

NFCRSH
IAQZERO
BADXOP
ERREX
CDUMP

NFCRSH is used by the microcode to cause a crash. It assumes that the MDO contains the crash code. The routine writes the crash code to >104 and then branches to >100 to crash. IAQZERO is used to end an instruction. It assumes that the next instruction address is mapped and that the PC points two beyond that. The routine reads the instruction into the IR and returns. BADXOP is used to signal an illegal XOP. It sets an illegal instruction interrupt and aborts. ERREX is used to take error returns. It assumes that R0 contains the error code and that the map and PC point to the address of the word containing the error return. The routine checks the error return and takes the correct action. This routine is similar to NFPOPO. CDUMP is used to dump the cache. It loads the MC from RF 15 (where a copy of the WP is kept) and dumps the cache. This routine destroys the MC.

All routines mentioned above except CDUMP are executed via the JUMP mnemonic. CDUMP is executed using the CALL mnemonic.

15.4.5 Debugging.

Most microcode debugging is done by code reading. The solution is usually obvious once the error is isolated. In cases where code reading of both assembly language and microcode does not isolate the error it may be useful to get a trace of the microcode. This can be done using a logic analyzer. If a logic analyzer is used, Table 15-1 shows where to connect the analyzer to the AU board of the system. The table only shows the location of the microaddress bus. To look at other signals, consult the schematic found in <u>Processors and Memories Vol. II,</u> Part Number 0945421-9702 *B.

Table 15-1 Location of the Microaddress Bus

Location	Pin	Function
Y5	8	Clock
X10	4	Ground
D10	8	UPC Bit 11
D10	7	UPC Bit 10
D10	6	UPC Bit 9
D10	5	UPC Bit 8
D10	4	UPC Bit 7
D10	3	UPC Bit 6
D10	2	UPC Bit 5
D10	1	UPC Bit 4
D11	23	UPC Bit 3
D11	22	UPC Bit 2
D12	21	UPC Bit 1
H8	12	UPC Bit 0

SECTION 16

DEVELOPMENT AND ANALYSIS TOOLS

16.1 OVERVIEW

The tools described in this section are for two purposes: DNOS development and DNOS analysis. The tools described first are shipped with DNOS. These include SRFI, TIGRESS, the System Debugger and PICT. The JENDAT editor is currently available only for Texas Instruments internal use.

16.2 SHOW RELATIVE TO FILE INTERACTIVELY UTILITY (SRFI)

The SRFI command displays and/or modifies the internal contents of files to VDT terminals. It assumes that the user has knowledge of the file structures involved. The file is accessed by physical records rather than logical records. The command is invoked by entering SRFI. Respond to the prompts as follows:

```
   SHOW RELATIVE TO FILE INTERACTIVELY
              FILE NAME:
           EDIT ACCESS?: NO
```

PROMPT DETAILS:

FILE NAME: The name of the file to be displayed/modified.

EDIT ACCESS?: Determines file access privileges. NO gives read only access, while YES gives exclusive write access. If no editing is desired, take the default NO response.

This utility works only on VDT terminals. The user is prompted for the record number by the utility. The utility starts with record 0 being displayed. The user may enter the record number desired in the field marked RECORD: 0000. If the record exists, it is displayed. If it does not exist, a >0030 error will be displayed as ERR: 0030. The F1 key moves forward through the file displaying data and updating the data displayed, while F2 moves backward through the file. If the entire record will not fit on the screen, the user may advance the data up through the data window by pressing the Previous Line key. The Next Line key moves the data down through the window.

To enter the edit mode press the F8 key. The cursor will move into the data fields at the upper left field of the data being displayed. The cursor can be moved through the data fields one at a time by pressing the Return key. This will advance the cursor to the next editable field. Previous Line will move the cursor up one line if it is available. Next Line will move the cursor down one line if it is available. Previous Field moves back one editable field. If the movement desired is not possible the cursor remains in the current field. The window will be moved across the file record as required to display large records. When the user moves the cursor through fields, the ASCII representation of the data will be modified to reflect the data that was in the field when the cursor left the field.

The data entered is not written to the file until the Enter key is pressed. SRFI then returns to the show mode on the same record that was being edited. The user may abort the update by pressing the Command key, which returns the user to show mode. The F1 and F2 keys also abort the edit before moving to the appropriate record in show mode.

Note that this utility provides a convenient means of recovering from >15 errors. Position the record containing the disk error on the screen, press F8 and then Enter. If the write was successful the error field will go to zero. Any data which may yet be invalid because of the >15 error may then be edited for correction.

It is also possible to edit the ASCII representation fields by entering the F7 key. This positions the cursor on the equivalent field in ASCII mode. F6 returns the cursor to the hexadecimal representations.

CAUTION

> The data returned is 7-bit ASCII. The high order bit will be returned as binary 0. The user may inadvertently alter data by editing in the ASCII fields.

16.3 THE TIGRESS TEST FACILITY

The Tigress program is used to establish an environment and exercise a test program using SVCs to the operating system. Tigress allows the user to define strings and values in Tigress task memory, to issue SVCs, to examine areas of Tigress task memory, and to use a file of data for input.

Tigress can be used interactively with commands input from a terminal, commands echoed to a terminal, and display information output to a terminal. Tigress can also be used in a batch stream with input and data from files or various devices.

Tigress is activated by direct task bid or by issuing the TIGR command to SCI. The command has the following prompts:

```
TIGR
           EXECUTE TIGRESS
     COMMAND ACCESS NAME: [pathname@]
     LISTING ACCESS NAME: [pathname@]
        DATA ACCESS NAME: [pathname@]
        ECHO ACCESS NAME: [pathname@]
                 TASK ID: integer          (A5)
              FIRST LUNO: integer          (0A0)
     2ND TIGRESS NEEDED?: yes/no           (no)
     STOP ON END ACTION?: yes/no           (yes)
  BUFFERED COMMAND ECHO?: yes/no           (no)
```

PROMPT DETAILS:

COMMAND ACCESS NAME

This prompt requests the source of the commands to be issued to Tigress. If a file name is used, input will be from that file. If the response is null, the terminal issuing the TIGR command is assumed to be the source of Tigress commands.

LISTING ACCESS NAME

If a file name is specified, output from Tigress display and status commands are placed in the file. If the response is null, the terminal issuing the TIGR command is used for such listings. Messages output to the listing file are preceded by XXXX: where XXXX is the line number of the command causing the message to be printed. Messages generated by Tigress in response to I/O errors to any of its files are of the form XXXX:YY-msg where XXXX is the line number of the command that was being processed, YY is an I/O error code, and msg is the Tigress message describing the error situation. An error count is kept, and a maximum of 25 errors is allowed before Tigress terminates. If output is not desired, enter DUMY.

DATA ACCESS NAME

If a file name is specified here, that file can be used for data input using the third LUNO of the set assigned by the TIGR proc.

ECHO ACCESS NAME

When a command is parsed by Tigress, the command is echoed for the user to see. If a file name is specified for this prompt, the echo output goes to the file. If the response is null, the echo goes to the terminal issuing the TIGR command. This feature is especially useful if commands are coming from a file and the user wants to watch the progress of the Tigress test at a terminal. If output is not desired, enter DUMY.

TASK ID

The task ID specified is that of the Tigress program to be used for the current tests. If you wish, you may put several versions of the program in the system program file, each with specific attributes needed for tests.

FIRST LUNO

Tigress uses four LUNOs to access the files or devices specified for COMMAND ACCESS NAME, LISTING ACCESS NAME, DATA ACCESS NAME, and ECHO ACCESS NAME. The LUNOs are four sequential numbers, beginning with the FIRST LUNO prompt response. These LUNOs are assigned by the TIGR proc before activating Tigress and released by the TIGR proc after Tigress terminates.

2ND TIGRESS NEEDED

If the Tigress task is to bid a second copy of Tigress, using its own access name, enter YES. Otherwise, enter NO. A YES response will bring up a second screen of prompts for access names and task ID for the second copy of Tigress.

STOP ON END ACTION

If YES is entered, Tigress will terminate if it encounters an error that causes it to go to end-action. If NO is entered and Tigress goes into end-action, it sets the command and list access names to be ME and processing resumes with commands from the terminal.

BUFFERED COMMAND ECHO

If buffered commands are to be executed with echo output, enter YES. This is most useful when debugging a new command file.

16.3.1 Details of Tigress Commands.

Table 16-3 shows the set of Tigress commands and their required parameters. Table 16-1 shows the argument types used in describing the commands.

Table 16-1 Types of Arguments for Tigress Commands

Type	Meaning
Address	Number or expression whose value is a valid Tigress task address
Number	Decimal value whose first digit is non-zero, hexadecimal value preceded by 0 or >, or an expression whose value is numeric
String	Alphanumeric characters surrounded by quote marks (")
Label	1 to 6 ASCII characters

An expression is any combination of numbers or addresses using addition or subtraction. The addresses used in an expression may be stated as labels that have been previously defined to have a numeric value. Arguments to the commands may also use signed numbers, preceding the number by a plus (+) or minus (-) sign. Indirect arguments can be specified by preceding the argument with an asterisk (*). In addition to labels defined by the user with the EQU command, there are a number of predefined labels. Table 16-2 lists and defines the predefined labels.

Table 16-2 Predefined Labels for Tigress Commands

Label	Meaning
ADDRND	First illegal (upper) Tigress address
CMDSCB	Address of SVC block used to read commands
DATSCB	Address of SVC block used to read data file
ECHSCB	Address of SVC block used to write echo
LSTSCB	Address of SVC block used for listing file
NEWMEM	Start of memory acquired with GET command
PC	The record number of the current command
R0 - R15	Tigress user registers 0 through 15
SVCB	Address of start of the SVC block used by Tigress to issue user specified SVCs
STK	Address of stack of arguments from last BRL command

The labels in Table 16-2 can be used in the DSP command to find the location of special Tigress structures. These labels cannot be redefined by the user.

The address space used by TIGRESS looks as follows:

```
          +-----------------------------------+
0000      |  REGISTERS R0 - R15               |
          |                                   |
          +-----------------------------------+
0032      |  SERVICE CALL CONTROL BLOCK       |
          |                                   |
          +-----------------------------------+
   ?      |  USER DEFINABLE ADDRESS SPACE     |
          |                                   |
          \/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\

          \/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
          |                                   |
          +-----------------------------------+
   X      |  FIRST INVALID ADDRESS            |
```

The entire set of command options is listed in Table 16-3. In addition to these, users may specify a comment line in a command file by placing an asterisk in the first column of the line.

The 3 letter command must begin in column 1 and the argument list must begin in column 5. Each argument, except strings, generates 2 bytes of data. The commands are described in the paragraphs following the table.

Table 16-3 Tigress Commands

Command	Parameters
BRL	label [...number/address/string]
DSP	number,address
END	
EQU	string,number
GET	number
INC	address,number
INP	address,number1,number2
JMP	number
JPB	"label"
JPF	"label"
LBL	"string"
MSG	string
MVI	address,number1,number2[,number3...]
MVS	address1,number,address2
RBT	address,number
RTE	
RTN	
SBS	address,number
SBR	address,number
SBT	address,number
SCB	number1,[,number2]
SEI	address,number1,number2[,number3...]
SHI	address,number1,number2[,number3...]
SLI	address,number1,number2[,number3...]
SNI	address,number1,number2[,number3...]
SES	address1,number,address2
SHS	address1,number,address2
SLS	address1,number,address2
SNS	address1,number,address2
SVC	[number,...]

BRL [label,...string]

The branch and link to subroutine command saves the current value of PC, builds an argument list in the stack area, and branches to a specified subroutine. (See also RTN and RTE.) For example, BRL SUB1,>0400,@>100,A,B,23,"ABC" would cause subroutine SUB1 to be called with a stack as follows:

PC	Location of the BRL command within the command file
>0400	
@>100	Whatever 16 bit address this resolves to
A	Whatever this is equated to
B	Whatever this is equated to
23	
>4142	AB
>4320	C
14	The length of the argument list in bytes

The value of STK is changed to point to the length word (the 14 in the above example). The stack itself resides within the Tigress task area, but is not modifiable by the user. It is expected that a subroutine would pick up its arguments as follows:

```
EQU   "ARG1AD",STK-*STK
EQU   "ARG2AD",ARG1AD+2
 .
 .
EQU   "ARG1",*ARG1AD
 .
```

NOTE

(1) Subroutines may call other subroutines several levels deep if needed. However, the total stack area is 120 bytes. An error will occur if a BRL is attempted which will overflow the 120 bytes limit and the BRL will be ignored.

(2)Text strings may be passed as arguments. They are put directly in the stack with a blank at the end if the length would be odd. The subroutine must either be capable of determining the length or the length should be passed as a separate argument. Also, such strings will quickly consume the 120 bytes of stack space, so use this feature with discretion. Since a variable length string in the stack would make it inconvenient to find an argument which followed the string, it is recommended that only the last argument (if any) of a subroutine be such a string.

(3) The subroutine address (SUB1 in the above example) must have been previously defined with an EQU command before the subroutine can be called.

DSP number,address

This command displays the number of bytes specified, starting at the address specified. The address is rounded to an even numbered address if necessary. Memory is displayed in multiples of 8 words per line of output. Each line of output shows the memory address of the first word displayed, eight words of data in hexadecimal, and the same eight words of data in ASCII.

END

The END command is used to terminate execution of Tigress. A termination message is output, and control returns to the task that activated Tigress.

EQU string,number

This command is used to assign a value to a label. The label can then be used as a parameter or as part of an expression in other commands. The string specified is the label to be used, and the number specified (or expression that evaluates to a number) is the value retrieved whenever the label is used.

GET number

This command gets 32 times the number of bytes of memory to be included in the Tigress address space. The address of the first word of this block of memory is equated to the string NEWMEM. Note that NEWMEM is undefined until this command is executed. On subsequent executions of the command, NEWMEM points to the start of the new block of memory acquired.

INC address,number

The number specified is added to the current value at the address specified.

INP address,number1,number2

This command causes number1 bytes of data to be read from the data file into memory at the address specified, starting from record number2 of the file.

JMP number

This causes Tigress to execute next the command that is located at a distance specified by number. JMP 0 indicates that this same command should be executed. JMP -2 indicates that Tigress should proceed to the command preceding this command by two, JMP 3 indicates that the next two commands should be skipped. This command is not meaningful if command input is from a terminal. (Comments count as commands.) JMP is an unconditional command but can be used with the skip commands below to accomplish branching.

JPB "label"
The jump backward to label causes a jump back in the command file until an LBL command is found with the same string operand as is on the JPB command. For example, JPB "ABC" jumps back in the file until a command of LBL "ABC" is encountered. The operand must be in quotes since it is not entered in the symbol table. It is treated as an ASCII string. The string may be of any length, but only the first six characters are used in the comparison.

JPF "label"
The jump forward to label command works like JPB except that the jump is in the forward direction rather than backward through the command file.

LBL "string"
This command denotes a place for a destination of a JPF or JPB command. The string may be from 1 to 6 characters long.

MSG string
The string specified is output to the listing file. This is useful for noting success or failure in a test stream and for documenting the test in progress.

MVI address,number1,number2[,number3...]
This command causes a move to the address specified of number1 bytes of data specified in number2 through the last parameter. Each argument creates two bytes of data. Example: MVI ADDR,1,>FF will move zero not >FF since the data value is >00FF.

MVS address1,number1,address2
This command causes a move to address1 of number1 bytes from address2.

RBT address,number
This command resets (sets to zero) the bit at the position specified by the number argument in the word at the specified address.

RTE

The return with error command returns from the current subroutine to the second command after the BRL which was last encountered. The stack is popped so that the value of STK is changed to the location of the saved argument list for the previous subroutine call (if any).

RTN

The return from subroutine command returns from the current subroutine to the first command after the BRL which was last encountered. As with RTE, the stack is popped.

SBR address,number

The SBR command tests the bit at the given position number in the specified address and skips the next command if the bit is reset (equal to zero).

SBS address,number

The SBS command tests the bit at the given position number in the word at the specified address and skips the next command if the bit is set to one.

SBT address,number

This command sets to 1 the bit at the given position number in the word at the specified address.

SCB number1[,number2]

This command builds the SVC at the address specified by the first argument, with the argument list beginning at the second argument address, and executes the SVC. If only one argument is specified, then the currently existing SVC at that address is executed. (This is similar to the SVC command except that the address is specified instead of defaulting to location SVCB)

SEI address,number1,number2[,number3...]

SHI address,number1,number2[,number3...]

SLI address,number1,number2[,number3...]

SNI address,number1,number2[,number3...]

Each of these commands compares the string of length number1 at the address specified with the value expressed in number2 through the last argument. If the comparison succeeds, the next command is skipped. SEI causes a skip if the comparison is equal, SHI if the first argument is high, SLI if the first argument is low, and SNI if the first argument is not equal to the immediate data. These commands are not meaningful if command input is from a terminal.

SES address1,number,address2

SHS address1,number,address2

SLS address1,number,address2

SNS address1,number,address2

Each of these commands compares the number of bytes specified at address1 with the data at address2. If the comparison succeeds, the next command is skipped. SES causes a skip if the comparison is equal, SHS if the first argument is high, SLS if the first argument is low, and SNS if the arguments are not equal. These commands are not meaningful if command input is from a terminal.

SVC [number,...]
If the argument list is null, the SVC currently at address SVCB is executed. If an argument list is presented, it replaces the data currently at SVCB, and the new SVC specified at SVCB is executed.

16.3.2 Directives of Tigress.

There are two directives which can be used to modify the manner in which Tigress reads and processes commands. These directives allow for buffering of Tigress commands to reduce file I/O interference for test instructions and test data. Like commands they consist of 3 characters and must begin in column 1 with column 4 left blank. Columns 5 through 80 may contain comments and are ignored. Directives are treated as commands and comments by the jump and skip commands. The directives are as follows:

BUF
Upon encountering this directive Tigress proceeds to read and buffer commands in its memory space. Approximately 200 commands may be buffered.

UNB
Upon encountering this directive Tigress discontinues buffering commands and begins executing commands in memory beginning with the first command buffered.

If a jump is made from a buffered command to a command outside the range of buffered commands the buffered mode is discontinued. If a jump is made into an area of buffered commands buffering mode is not initiated because Tigress did not encounter the BUF command. A BUF command following a BUF command but preceding the UNB command causes no action. An UNB command not preceeded by a BUF command causes no action. While in buffering mode any messages output by the MSG command will also be buffered. If MSG commands are included in a BUF-UNB block of commands then the number of commands which may be buffered is reduced. Echo output while in buffer mode is suppressed. The DSP command output is not affected by the buffer mode of operation.

16.3.3 User Defined Commands For Tigress.

When Tigress encounters a command it searches its definition table and if the command is found a BLWP is made with the value of the command in the table being the address of the start of the code to be executed for that command.

This method of implementing command execution enables the user to define his own command as if if it were a Tigress command. The procedure for doing this is to equate a 3 character label to a location at which code is to be executed. By using MVI commands the machine code is placed in memory. The last machine instruction must be a RTWP (>0380). When the code is to be executed simply enter the label equated as a command. Tigress will search the table of equates and, finding the label, will process it as a command. This feature can be used for things such as initiate mode I/O where a separate call block must be built and executed.

16.4 THE SYSTEM DEBUG UTILITY

Since it is not always possible or convenient to use the SCI debugger or some other testing task, an interactive system debug program is available. This program allows the user to display and/or modify the workspace pointer, program counter, registers, and memory; to set as many as 16 software breakpoints in the code being debugged, and to step through the code one instruction at a time. The program interacts with the user by doing I/O to a 911 VDT or an ASR device. This program does not work on a 940 or 931 VDT.

About twenty-five commands are available for use with the debug program. Each command is specified by a single character. When the character is recognized as a command, the debug program calls the appropriate processor, that may in turn require additional parameters to be input. Whenever the program is expecting a command to be input, it prompts with a question mark (?). Parameters that are entered are usually hexadecimal constants of up to four digits. A parameter is terminated by the fourth digit or by a period or Return key if less than four digits are specified. If an invalid digit is typed, the entire parameter is ignored and may be reentered for most commands. In some cases, invalid input causes the debug program to terminate.

16.4.1 Details of Debug Commands.

The entire set of commands and parameters is shown in Table 16-5. Each command is detailed in the paragraphs that follow. Commands that allow the user to modify contents of some location first display the current value. If no value is entered, no change is made. Table 16-4 shows the command parameter types and their meanings. Brackets indicate optional parameters.

Table 16-4 Parameter Types for Debug Commands

Type	Meaning
rel	Offset relative to the current base address
adr	Absolute address in a task
bt	Beet bias address for the segment being addressed, located via map file information carried in the TSB

NOTE

Breakpoints should not be used on instructions that change CURMAP (for example, instructions that move something to the OS label CURMAP) or on instructions that change map file 0 (for example, LMF Rn,0). They must never be set on interruptable instructions; that is, XOP, MOVS, and a number of /12 instructions.

Table 16-5 Commands for System Debug Program

Command	Description
A adr1[bt adr2]	Display and alter memory long distance
B rel	Set a breakpoint relative to current base
C	Display condition code in status register
D adr1 adr2[bt adr]	Display memory long distance
E rel1[rel2]	Examine locations relative to current base
F	Display and alter following memory long distance
I [adr1 adr2]	Inspect local memory
J	Show all local workspace registers
L	List all instruction breakpoints
M adr	Display and alter contents of memory address
N	Display and alter contents of next memory address
O	Set address for base of relative offsets
P	Display and alter current program counter
Q	Quit debugging session
Rn	Display and alter contents of register n (Hex)
S adr	Set breakpoint at address in local address space
U [adr]	Unset one or all breakpoints
W	Display and alter address of workspace pointer
X	Execute a single instruction and stop
Z	Continue after current breakpoint
+ n m	Add the numbers n and m
- n m	Subtract the number n from m
?	Display a menu of all commands

A - Display and alter memory long distance

This command is used to display and alter memory that is not mapped into the current task address space. The form of the command is A xxxx bbbb yyyy where xxxx is the address of the memory location to be displayed and altered and bbbb is the starting beet bias of the segment in which the memory 'location resides. The optional yyyy parameter specifies a logical address at which the segment bbbb is supposed to start. The appropriate beet bias can be determined from the map file information in the task status block (TSB) or the segment status block (SSB) of the task being debugged. The bbbb and yyyy values default to their previous values.

B - Set a breakpoint relative to current base
This command is valid only if the base address has been set correctly using the O command or the base address of the program being debugged is zero. The form of the command is B xxxx where xxxx is the offset from the base value at which a breakpoint should be set. When the program is executing and reaches the breakpoint, all current workspace registers are displayed; and the debug program prompts for a debug command.

C - Display condition code in status register
The form of this command is C=xxxx yyyy where xxxx is displayed as the current contents of the status register and yyyy is entered by the user as the new status register contents.

D - Display memory long distance
This command is used to inspect a portion of memory that is not mapped in the current task address space. The form of the command is D xxxx yyyy bbbb zzzz where xxxx is the starting address of the to memory to be displayed, yyyy is the ending address, and bbbb is the starting beet bias of the segment in which this portion of memory resides. The bbbb and zzzz default to their previous values. The optional zzzz specifies the starting logical address of the segment at bbbb. The beet bias can be retrieved from the task status block (TSB) of the task being debugged (use TSBML1, TSBML2, or TSBML3 depending on whether segment 1, 2, or 3 is being examined), or use the SSB to get the address. Like the I command, this command displays memory in blocks of >10 bytes, showing the starting address of each line of the display.

E - Examine locations relative to current base
This command is valid only when the base address has been set using the O command or the starting address of the program being debugged is zero. The form of the command is E xxxx where xxxx is an offset from the current base. The >20 bytes of memory starting at the offset is displayed in the same format as used for the I command.

F - Display and alter following memory location long distance
This command can be used after an A command to examine the following memory location, similar to the N command is used after an M command.

I - Inspect local memory

This command allows the user to inspect a range of memory locations in the local task address space. The form of the command is I xxxx yyyy where xxxx is the starting address of the range to be inspected and yyyy is the ending address of the range. If an invalid address is specified, the debug program will terminate.

Both the starting address and ending address are optional. If neither is specified, >20 bytes of memory are displayed, starting at the current program counter address. The display shows >10 bytes per line of the display, preceding the data with the first address of the data. If only the starting address is specified, >20 bytes of memory are displayed, starting at that address. If both arguments are supplied, a multiple of >10 bytes is shown, starting at an even address and including at least the amount specified.

J - Show all local workspace registers

This command displays all workspace registers on one line of the display.

L - List all instruction breakpoints

This command lists all breakpoints currently set in the program. If an initial base address has been set using the O command, this list includes the relative offset from the base for each breakpoint as well as the absolute address of each breakpoint.

M - Display and alter contents of memory address

This displays M xxxx=yyyy zzzz where yyyy is the current value at local memory address xxxx and zzzz is the desired new value at that address. Only addresses currently mapped in the task space may be modified with this command.

N - Display and alter contents of next memory address

This command is used after an R, M, or another N instruction to display and/or alter the next memory address after the address examined previously. The command form is N xxxx=yyyy zzzz where xxxx is the address displayed by the debug program, yyyy is the current value, and zzzz is the new value supplied by the user.

O - Set address for base of relative offsets

The form of this command is O xxxx yyyy where xxxx is supplied by the debug program to show the current base and yyyy is supplied by the user as the new base. After this command has been used to specify the initial address of a program, the B and E commands can be used to set breakpoints and examine memory locations according to offsets from that base. This feature enables a user to work from an assembly language listing and easily determine where to stop execution and where to examine data.

P - Display and alter program counter
This displays PC=xxxx yyyy where yyyy is typed by the user to indicate the new value desired for the program counter.

Q - Quit debugging session
The Q command terminates the debug program. If the debug program is linked as part of the operating system, this causes the system to idle. If linked with a user task, the Q command terminates that task.

Rn - Display and alter register n
This displays Rn=xxxx yyyy where yyyy is typed by the user to indicate the new value desired in register n.

S - Set a breakpoint
With this command, the user can set a breakpoint at any address currently mapped in the task space. The form of the command is S xxxx where xxxx is the local memory address at which execution should halt. When the breakpoint is reached, all current workspace registers are displayed; and the debug program prompts for a debug command.

U - Unset one or all breakpoints
The form of this command is U xxxx where xxxx is the address at which a breakpoint has been set by either the S or the B command. The breakpoint specified is unset. If xxxx is not specified, all currently set breakpoints are unset.

W - Display and alter workspace pointer
This displays WP=xxxx yyyy where yyyy is typed by the user to indicate the new value desired for the workspace pointer.

X - Execute a single instruction
This command allows the user to step through the program being debugged, one instruction at a time. After one instruction is executed, all current workspace registers are displayed; and the debug program prompts for a debug command. This command may not work predictably when used on a breakpoint which is currently set.

Z - Continue after breakpoint
When this command is issued, execution proceeds from the current breakpoint to the next breakpoint, if there is any. Unless a U command is used, the breakpoint that was just used remains in the code and can be reached again.

+ - Add numbers
The form of this command is + xxxx yyyy zzzz where the user types xxxx and yyyy and the debug program computes zzzz as xxxx+yyyy.

- - Subtract numbers
  The form of this command is - xxxx yyyy zzzz where the user types xxxx and yyyy and the debug program computes zzzz as yyyy-xxxx.

? - Display a menu
  This displays the menu of all available debug commands, showing the characters required and an English description of the commands.

16.4.2 Establishing the Debug Environment.

The system debugger can be added to the disk image of a DNOS system by executing the DEBUG command, located in the .S$SYSTEM.S$$CMDS command library. Before executing the DEBUG command, the following preparations should be made:

1. Verify that the DEBUG procedure is in .S$SHARED as procedure ID 1.

2. Make the two root segments on the image program file updateable. This can be done by issuing the XSCU and QSCU commands. Issue the XSCU command with the following responses.

```
[] XSCU

EXECUTE SYSTEM CONFIGURATION UTILITY

              SYSTEM VOLUME:  <volume name>
                SYSTEM NAME:  <system name>
```

where:

An LDC listing appears. Press the Command key.

Now issue the QSCU command with the following response:

```
[] QSCU

QUIT CONFIGURATION UTILITY SESSION

                     ABORT?:  NO
```

3. Determine the illegal XOP WP and PC values by issuing the LSM command with these responses:

```
[] LSM

LIST SYSTEM MEMORY

      OVERLAY NAME OR ID:  ROOT
        STARTING ADDRESS:  >50
         NUMBER OF BYTES:  040
     LISTING ACCESS NAME:
```

   Record the first two values starting at address 0050 (for example, 2CA8, C56A). These are the illegal XOP WP and PC values, respectively.

4. Now you are ready to install the debugger. Enter .USE to give access to the DEBUG command ([ ] .USE S$SYSTEM.S$$CMDS, .S$CMDS).

5. Then enter the DEBUG command:

```
[]  DEBUG

ADD/REMOVE SYSTEM DEBUGGER

    ADD/REMOVE/MODIFY?:  alphanumeric
    TARGET DISK/VOLUME:  device name@                 (*)
           SYSTEM NAME:  alphanumeric                 (*)
  XOP LEVEL ( 0 - 14 ):  integer                      (1)
   DEBUG TERMINAL CRU:  integer                   (>100)
   DEBUG TERMINAL TYPE:  {911,KSR,ASR,VDT,EIA,TTY} (911)
     ILLEGAL XOP    WP:  integer                  (>2C48)
     ILLEGAL XOP    PC:  integer                  (>C56A)
```

   Use the debug command to add or remove the system debugger from the disk image of a DNOS system, which is located in program file <target disk/volume>.<system name>. The command may also be used to modify the operating characteristics (for example, debug terminal CRU address) of a previously installed system debugger.

   PROMPT DETAILS:

   ADD/REMOVE/MODIFY?
      Enter ADD to add the debugger to a system. Enter REMOVE to delete the debugger from a system. Enter MODIFY to change the operating characteristics of a previously added debugger.

TARGET DISK/VOLUME
Enter the name of the disk volume containing the system to be debugged.

SYSTEM NAME
Enter the name of the system to be changed. The system kernel image file resides on a program file with the name <TARGET DISK/VOLUME>.<SYSTEM NAME>.

XOP LEVEL [0 - 14]
Enter the XOP level to be used by the debugger as a breakpoint instruction. The default is 1, but if the XOP level 1 is already in use, another level between 0 and 14 may be selected.

DEBUG TERMINAL CRU
The system debugger uses direct CRU I/O to terminal. Enter the CRU address of the terminal where you want debugger output to go.

DEBUG TERMINAL TYPE
Enter the type of terminal the debugger will write to. Choices are 911, VDT, TTY, EIA, ASR, KSR. The ASR or KSR must be connected through an EIA interface module.

ILLEGAL XOP WP
Enter the WP value recorded from the earlier LSM.

ILLEGAL XOP PC
Enter the PC value recorded from the earlier LSM.

Messages:
The DEBUG command will produce a listing in the terminal local file which shows the XOP instruction to use for a breakpoint, the CRU address, and the terminal type.

Notes:
If the XOP level default is not selected, the normal breakpoint instruction (>2C40) set by the B instruction, will change.

6. Note the third word of the display returned by the DEBUG command. This is the value which will be used as the breakpoint instruction. It should be equivalent to the assembly language instruction

```
XOP R0,<xop level>
```

where <xop level> is the number you entered for the fourth prompt of the DEBUG command. Using listings and linkmaps, choose the address of the task (DSR, SVC processor, etc.) where you wish to set your initial breakpoint. Use the MPI (or MRF) command to modify the executable code image at that address to contain the value noted above. Remember the proper value so that the instruction can be be restored when you are debugging.

7. Reboot the system or execute the task or whatever it takes to cause entry to the code of interest at the selected address. Use the M debug command to restore the code at the initial breakpoint address, then set up the debug environment that you require. Use the X or Z debug command to proceed with the execution of the code to be debugged.

16.5 THE PICT UTILITY

The PICT utility is used in the DNOS source library to create tables from the assembly language templates to be used by Pascal code and to generate documentation pictures of the assembly input, generating Pascal record descriptions and/or line drawing picture files. PICT is controlled by the use of macros (verbs) in the assembly language input files. These macros expand to appropriate assembly language code when used with the macros defined in the DNOS file .S$OSLINK.MACROS.TEMPLATE directory.

The PICT utility can be accessed by using the PICT command in the .S$SYSTEM.S$$CMDS directory. The command has the following prompts, with descriptions as outlined below.

```
      PICT (CREATE DATA TABLE PICTURE)

         SOURCE FILE(S): filename
           PICTURE FILE: [filename]
     PASCAL OUTPUT FILE: [filename]
          PAGE CONTROL?: (YES,NO,Y,N)  (YES)
```

Prompt Details:

SOURCE FILE(S)

Specify the file or files you want to have examined by the picture processor. If more than one file is specified, the files are concatenated and processed as a single file.

PICTURE FILE

Specify the pathname for the picture file to be created

PASCAL OUTPUT FILE
Specify the pathname for the file of equivalent Pascal statements.

PAGE CONTROL?
Specify NO if you want no embedded carriage control in the picture file that is being built. Specify YES if you want carriage control. If you specify YES, the picture file will have a notation of (CONTINUED) after every 55 lines.

The complete set of verbs available with PICT is shown in Table 16-6 along with the intended purpose of each verb.

The verbs are shown with brackets [] to indicate optional arguments and braces {} to indicate that a choice must be made from the indicated options.

Since the assembly language macros automatically generate the label xyzSIZ for the structure named xyz, users must avoid using their own labels of the same format.

The verbs are used in appropriate groupings to define various types of structures. The major structures are framed by DORG and RORG, CSEG and CEND, and PCKREC and ENDREC. The first pair of verbs is used to to define an assembly language DSEG and a corresponding Pascal packed record variable declaration. The second is used to define an assembly language CSEG (with data to be initialized in an assembly language routine) and a corresponding Pascal packed record variable declaration. The third pair of framing verbs provide an assembly language DSEG and a Pascal packed record type declaration. The first and second set of framing verbs are intended to be close to the work done in assembly language, while the third set provides easy definition of Pascal packed record structures with variants.

Table 16-6 Verbs Used in Generating Structures

	VERB		
[label]	ADDR)	0	
[label]	ARRAY	number of elements, {INT,LONG,POSINT,WORD} or a type defined with PCKREC	[comment]
	BITS	[label,] number of bits	[comment]
[label]	BSS	number of bytes	[comment]
[label]	BYTE	0	[comment]
	CEND		
[label]	CHAR	number of characters	[comment]
	COPY	filename	
	CSEG	'label'	
[label]	DATA	0	[comment]
	DORG	n or label	[comment]
	ECHO		
	ENDREC		[comment]
label	EQU	n or label	[comment]
[label]	EVEN		[comment]
	FLAG	label	[comment]
[label]	FLAGS	{8,16}	[comment]
[label]	INT	0	[comment]
	LIST		
[label]	LONG	0	[comment]
	PAGE		
	PCKREC	label	[comment]
[label]	POSINT	0	[comment]
[label]	PTR	type	[comment]
[label]	REC	type specified by PCKREC above	[comment]
	RORG		
	UNL		
	VARNT	n or label	[comment]
[label]	WORD	0	[comment]

The COPY verb is used to bring in a file before the PICT utility processes the entire input file. If the input file refers to a user-defined type or constant, the appropriate file must be copied in to supply the definition. Any number of COPY verbs may be used, but they cannot be nested. That is, a file may not copy in a file which uses the COPY verb.

Most of the other verbs can be used independently of each other, and they can appear in any of the three framing verb pairs. An exception is the set of verbs used to define flags fields. The verbs FLAGS, FLAG, and BITS must be used in a relatively restricted fashion. The FLAGS verb must appear first, defining the number of flags being generated to be either 8 or 16. This can be followed by an appropriate number of FLAG and or BITS verbs to complete the field of 8 or 16. The entire field does not need to be explicitly defined. PICT will generate a filler label and allocation in the Pascal structure, and the assembly language macros generate only the required equates for the flags defined.

For many of the verbs, the operand field is optional. However, if a comment is used, the operand field must be supplied to avoid parsing part of the comment as operand.

16.5.1 Assembly Language Output.

The following paragraphs describe the effect of each of the verbs for the assembly language output. The succeeding set of paragraphs describe the Pascal output, and a third set of paragraphs describe the picture output generated by PICT.

[label] ADDR 0
    This generates a one word integer

[label] ARRAY n,type
    This generates an optionally labeled field with a BSS for the appropriate number of bytes to allocate n instances of the type specified.

BITS [label,]n
    If a label is specified, an EQU is generated with the label equated to the current autogenerated bit number within the FLAG field.

[label] BSS 5
    This is a standard assembly language directive.

[label] BYTE 0
    This is a standard assembly language directive.

CEND
    This is a standard assembly language directive.

[label] CHAR n
    This generates a BSS for the number of characters specified.

COPY filename
    This is a standard assembly language directive.

CSEG label
    This is a standard assembly language directive.

[label] DATA 0
    This is a standard assembly language directive.

DORG n or label
    This is a standard assembly language directive.

ECHO
    This is ignored.

ENDREC
    This terminates a record definition begun with PCKREC. It generates a size equate aaaSIZ where aaa is the name of the packed record and an RORG 0 statement.

label EQU n or label
    This is a standard assembly language directive. RESERVE BLOCK 3

[label] EVEN
    This is a standard assembly language directive.

FLAG label
    This generates an EQU, with the label equated to the current autogenerated bit position within the FLAGS field. The first flag position is always position 0.

[label] FLAGS {8,16}
    If the operand is 8, this generates a BSS 1. IF the operand is 16, it generates a BSS 2.

[label] INT 0
    This generates a BSS 2.

LIST
    This is a standard assembly language directive.

[label] LONG 0
    This generates a BSS 4.

PAGE
    This is a standard assembly language directive.

PCKREC name
    This begins a packed record, indicated by a DORG 0. Notice that if the packed record is to be assembled as well as used with PICT, the structure name is limited to a maximum of three characters.

[label] POSINT 0
    This generates a BSS 2.

[label] PTR type of pointer
    This generates a BSS 2.

[label] REC type
    This uses the aaaSIZ equate built during processing of a PCKREC to generate a BSS of the appropriate size.

RORG
    This is a standard assembly language directive.

UNL
    This is a standard assembly language directive.

VARNT n or label
    This generates a DORG n, where n is the supplied value or the value of the label.

[label] WORD 0
    This generates a BSS 2.

16.5.2 Pascal Template Output.

The following paragraphs describe the output generated by PICT for use in Pascal code. In cases where a label is output to the Pascal file but no label is supplied by the input line, PICT generates a label of the form FILLxy where xy begins at 00 and is incremented by 1 with each new filler label used.

In most cases, the comment found on an input line which generates an output line is also found on that output line. The exceptions are output lines of PACKED RECORD, CASE INTEGER OF, and variant labels.

Each output file is intended to be unlisted in a Pascal program. The first line is always (*$ NO LIST *) and the last line is always (*$ RESUME LIST *).

[label] ADDR
    This generates: label : ADDRESS; (ADDRESS is defined in the DNOS file .TEMPLATE.PTABLE.TYPES as 0..#FFFF)

[label] ARRAY n,type
    This generates: label : PACKED ARRAY [1..n] OF type;

BITS [label,]n
    This generates: label : 0..m; where m is the maximum value which can be expressed in n bits. For example, BITS ALPHA,3 generates ALPHA : 0..7;

[label] BSS n
    This generates: label : PACKED ARRAY [1..n] OF BYTE;

[label] BYTE 0
    This generates: label : BYTE; (BYTE is defined in the DNOS file .TEMPLATE.PTABLE.TYPES as 0..#FF)

CEND
    This generates: END; for a CSEG file.

[label] CHAR n

This generates: label : PACKED ARRAY[1..n] OF CHAR;

COPY filename

This causes no Pascal output.

CSEG label

This is the beginning of a packed record named by the CSEG label. It generates: label = PACKED RECORD (where the label has no quotes, though the CSEG label does)

[label] DATA 0

This generates: label : WORD; (WORD is defined in the DNOS file .TEMPLATE.PTABLE.TYPES as 0..#FFFF)

DORG n or label

Depending on where it appears in an input file, this might be the start of a packed record or the beginning of a variant in the packed record. It is recommended that PCKREC be used when creating new structures and that DORG be used only for compatibility purposes. (DORG will also be needed if the structure must have a starting location counter value that is non-zero.)

When encountered the first time in a file, DORG 0 generates xxx = PACKED RECORD where xxx is the first three characters in the next line with a label (unless that line has an EQU directive, in which case it is skipped). When encountered in succeeding lines of the file, DORG generates a variant at the current level. Thus several DORG statements in succession with the same operand will generate variants at the same level. A new operand on a succeeding DORG defines a deeper level of nesting. It is necessary, of course, to have all variants (and variants within variants) at the end of the structure being defined. (See the description of VARNT for further details.)

ECHO

The entire input file is ignored and no Pascal output is generated other than the NO LIST and RESUME LIST directives. The ECHO option is intended for files of equates needed only for assembly language use.

ENDREC

This generates: END; for the packed record under construction.

label EQU n or label

This generates no Pascal output.

[label] EVEN

This is ignored.

[label] FLAGS {8,16}
This begins a flags field, that is a packed record of boolean values. It generates: label : PACKED RECORD

FLAG label
This identified a flag and generates: label : BOOLEAN;

[label] INT 0
This generates: label : INTEGER;

LIST
This is ignored.

[label] LONG 0
This generates: label : LONGINT;

PAGE
This is ignored.

PCKREC name
This generates: name = PACKED RECORD

[label] POSINT 0
This generates: label : POSINT; (POSINT is defined in the DNOS file .TEMPLATE.PTABLE.TYPES as 0..#7FFF)

[label] PTR type of pointer
This generates: label : @type;

[label] REC type
This generates: label : type;

RORG
This is used to terminate a packed record started by DORG 0. It generates END;

UNL
This is ignored.

VARNT n or label

This begins a variant of the current packed record at the current level if the operand is the same as the start of the current level. If the operand is not the same, a next level of variant is begun. This requires that variants be defined in correct order of nesting. That is, variants of a structure (or of variants) appear at the end of the structure. At the first VARNT of a given level, the following is generated:

```
CASE INTEGER OF
     1: (
```

Succeeding variants at the same level generate succeeding integer labels and open parentheses for the case statement. Variants of variants generate a CASE statement and the label 1: (for the first such variant and succeeding integer labels and open parentheses for the following variants. The termination of the structure (ENDREC, RORG, or CEND) cause the output of all required matching parentheses to close the CASE(s) currently open.

[label] WORD 0

This generates: label : WORD; (WORD is defined in the DNOS file .TEMPLATE.PTABLE.TYPES as 0..#FFFF)

16.5.3 PICT Picture Output.

The following paragraphs describe the output generated by PICT in its picture output file. Consult Figure 16-3 for an example of some of the output generated. Any structure that must output more than four words of unlabeled blocks outputs a broken picture but maintains an accurate location counter value. Fields that must start on word boundaries do so, with the picture showing an unlabeled byte preceding that word boundary.

Each line of the picture carries the associated comment of the input line, as well as portraying the space occupied by the verb in use on that line. Some input lines do not affect the picture but are used for information which appears after that picture. Flag details, equate information, and special comments follow the picture. A PAGE verb must appear at the end of the input file to cause output of flag details, equate information, and special comments.

[label] ADDR

This outputs a labeled block of one word on a word boundary.

[label] ARRAY n,type

This generates a labeled block of one byte (if the array occupies only one byte or begins on an odd byte boundary) or a labeled block of one word, followed by unlabeled blocks filling out the structure.

BITS [label,]n
This generates an entry in the equates listing at the end of the picture file, specifying the label and its location in the structure.

[label] BSS n
This generates a labeled block of one byte in the diagram and an appropriate number of unlabeled blocks.

[label] BYTE 0
This generates a labeled block of one byte.

CEND
This is ignored.

[label] CHAR n
This generates a labeled block of one byte and an appropriate number of unlabeled blocks.

COPY filename
This is ignored.

CSEG label
This is assumed to occur only at the start of the picture. Thus all initial conditions hold -- the location counter is zero, no output is generated.

[label] DATA 0
This generates a labeled block of one word on a word boundary.

DORG n or label
This sets the location counter to the operand value, terminates any picture in progress, outputs the comment on the DORG line, and sets conditions to start another picture segment (indicated by *----------+----------*).

ECHO
This causes the entire input file to be written to the picture file as it is read. It is assumed to be a file which otherwise generates no meaningful picture.

ENDREC
This finishes a picture section, drawing a line below any partially completed block and outputting the size of the packed record just completed. It also outputs the message **END OF PACKED RECORD.

label EQU n or label
This generates a line of output in the listing of equates

which follows the picture.

[label] EVEN
If the picture is currently not at a word boundary, an unlabeled block of one byte is output.

[label] FLAGS {8,16}
A labeled block of one byte is output if the operand is 8; a block of one word on a word boundary is output if the operand is 16.

FLAG label
This generates an entry in the flags descriptions which follow the picture. Each flag is shown with its relative position in the field as well as any comment describing that flag. Comments on lines following the FLAG input line are also echoed in the flags description in the picture output file.

[label] INT 0
This generates a labeled block of one word on a word boundary.

LIST
This is ignored.

[label] LONG 0
This generates two words, with the first word carrying the label and appearing on a word boundary.

PAGE
This is used to determine that the end of the structure has been reached. It causes flags and equate descriptions to be output. Any comment lines which follow the PAGE line will be output at the end of the picture listing under the heading COMMENTS ON THIS STRUCTURE. This verb is REQUIRED in order to output flags and equates information to the picture file.

PCKREC name
This begins a picture section, with the location counter set to zero. It outputs a line with **BEGINNING PACKED RECORD name.

[label] POSINT 0
This outputs a labeled block of one word on a word boundary.

[label] PTR type of pointer
This outputs a labeled block of one word on a word boundary.

[label] REC type
This outputs a labeled word on a word boundary and an appropriate number of unlabeled blocks to encompass the

total required by the type specified.

RORG
This is ignored.

UNL
This is ignored.

VARNT n or label
This finishes the current picture section, sets the location counter to the operand specified, and initiates a new section. It outputs any comment on the VARNT input line.

[label] WORD 0
This outputs a labeled block of one word on a word boundary.

16.5.4 Input Format.

The input file may be in one of several forms, one of which is shown in Figure 16-1 and another which is shown in Figure 16-2. The name of the structure is shown as aaa. The heading comments are of the form used for DNOS structures and are not enforced by the PICT utility. Any comments that precede the first structure verb appear in the output files before the structure. Comments to be printed at the end of the picture file must appear after a PAGE statement.

Figure 16-3 shows the format of the picture drawn by PICT. Each line of the drawing is preceded by the hexadecimal offset into the template. Each field carries its own label.

```
       UNL
*************************************************************
*                                                           *
*  <full structure name>      (<aaa>)                 <date>*
*                                                           *
*           LOCATION: <location in system>                  *
*************************************************************
<any comments to appear before the structure>
<any COPY statements needed by statements in this file>
        PCKREC name
<label> <verb>  <value>      descriptive comment
    .
    .
    .
<label> <verb>  <value>      descriptive comment
        ENDREC
<more packed records might be defined here>
        PAGE
        LIST
```

Figure 16-1 PCKREC Input Format

```
       UNL
*************************************************************
*                                                           *
*  <structure name>      (<aaa>)                      <date>*
*                                                           *
*           LOCATION: <location in system>                  *
*************************************************************
<any comments to appear before the structure>
       DORG <value>  or  CSEG 'label'
<label> <verb>  <value>      descriptive comment
    .
    .
    .
<label> <verb>  <value>      descriptive comment
aaaSIZ EQU  $
       RORG          or CEND
       PAGE
       LIST
```

Figure 16-2 DORG Input Format

```
    <all comments which preceded the first structure verb>

        *----------+----------*
  >00   !  <label> ! <label>  !  <comment>
        +----------+----------+
  >02   !       <label>       !  <comment>
        +----------+----------+

                <etc>
        +----------+----------+
  >mn   !       <label>       !
        *----------+----------*

 FLAGS FOR FIELD: <fieldname>  #mm - <label>
       <flagname> = (x... .... .... ....) - <comment>
                                            <special comments>
                                            <etc>
       <flagname> = (..x. .... .... ....) - <comment>
             <etc>

 <repeated for each flag field>

 EQUATES:
    FIELD   OFFSET  EQUATE  VALUE   DESCRIPTION
    -----   ------  ------  ------  ----------------------------
   <label>   #nn    <label>  #mm    <comment>
       <etc>
```

Figure 16-3 Template Picture Format

16.6 THE JENDAT EDITOR

The JENDAT editor is used to edit the JENDAT file used by DNOS system generation. It includes commands to show the current content of the JENDAT file, to edit any field, to format a file for printing, to remove records from the file, to change the version number of the file, and to exit the JENDAT editor.

16.7 XJENED Command Procedure

The DNOS JENDAT editor is invoked from SCI through the XJENED command procedure. No prompts are included. The program expects to find a value for synonym JENDAT, which specifies the file that the editor edits. The normal value of the synonym is .S$SGU.JENDAT. If the synonym is not defined, a file named .JENDAT@$ST is created (where $ST is a synonym for the users station ID). This file is empty and does not produce the desired results. This error is easy to detect. A SHOW or REMOVE command responds with NUMBER OUT OF RANGE in the error field. The EDIT command responds with NEW RECORD to any entry in the record number field. Quit the edit and assign the proper value to synonym JENDAT.

The program also expects a value for synonym PFILE, which specifies the file used for producing a formatted copy of the JENDAT file. This data goes to .PFILE@$ST otherwise. This file should be precreated with a logical record length of 132. If the file has a logical record length less than 132, the file will have twice the normal number of records. For example, the first line becomes two lines, the first containing the numeric data from the JENDAT record, the second containing the text portion. The file contains escape sequences that compress the print of the 810 printer and then restore it for normal operation at the end of the file. The file should be printed with 82 lines per page.

NOTE

> The DNOS JENDAT editor requires a 911 VDT. Otherwise, it will not function properly.

16.8 JENED Commands

The commands available for the maintenance of the DNOS JENDAT file are as follows:

COMMAND	MEANING
EDIT	Edit any field of a record)
PRINT	Format a file for printing)
QUIT	Exit the JENED editor)
REMOVE	Remove records from the JENDAT file)
SHOW	Show the text of the JENDAT records)
VERSION	Change the version number of the JENDAT file
MOVE	Relink records of the file

The JENED program begins with a display of the main menu, which lists the commands available to the user. The main menu is as follows:

```
              SELECT ONE OF THE FOLLOWING :

          P - PRODUCE A FILE FOR PRINTING
          V - CHANGE VERSION NUMBER OF JENDAT
          R - ZERO FIELDS OF JENDAT RECORDS
          S - SHOW TEXT OF JENDAT RECORDS
          E - EDIT FIELDS OF JENDAT RECORDS
          M - RELINK JENDAT RECORDS
          Q - QUIT
                       COMMAND : _
```

The character entered does not terminate the call. To process the command, the user must hit RETURN. Invalid entries are ignored.

16.8.1 EDIT Command.

When the EDIT command is entered, the menu field is replaced by the editing template. The following is an editing template:

```
                         EDIT RECORD NUMBER ____

.........1.........2.........3.........4.........5.........6.....

                DEF ____ AAT ____ LB ____ UB ____ NEXT ____
```

The number entered is a decimal number. A zero entry or empty entry places the editor in the append mode. As a result, the first field indicates that a new record is being entered. The first field is modified as follows:

```
       EDIT RECORD NUMBER ____          NEW RECORD
```

The text field is initialized with blanks. The first four numeric fields are initialized with 0000, and the NEXT field is initialized to the record number of the record following the current record.

If an error is detected in the field, the following error message is displayed:

```
                             EDIT RECORD NUMBER 12AE
       ERROR IN NUMERIC FIELD
```

If the record number is larger than that of the last record of the current file, the editor enters the insert mode. The record displayed is one larger than the last record, and the message NEW

RECORD is displayed. If the record number exists, the corresponding data appears on the screen. The cursor is positioned in the first column of the text field when a valid record number has been chosen.

The JENDAT editor requires a call termination character for each read that is issued. The valid termination characters, which vary with the call that is outstanding, are as follows:

* CMD

* RETURN (SKIP)

* ENTER

* Up Arrow

* Down Arrow

* F1 (function key 1)

* F7 (function key 7)

* F8 (function key 8)

* TAB

* Left Field

CMD Key

The CMD always returns the user to the main menu from the edit mode, whether editing existing records or inserting new records. The current record is never updated with the data on the screen.

RETURN Key

The RETURN key causes the call to progress from the text field to the DEF field, then to AAT, LB, UB, and NEXT, in that order. When the RETURN key is entered from the NEXT field, the data on the screen is moved to the file. The next physical record is entered on the display, and a read is issued to the text field. The record number field is updated to the record number of the data that is displayed. The SKIP key causes the same effect as RETURN but blanks out the remainder of the field. This produces acceptable results in the record number field and the text field but should be avoided in the numeric fields.

NOTE

> The JENDAT file is not a forced write file. Therefore, the data is moved to the memory buffer that will be written by file management when necessary to free buffer space.

ENTER Key

The ENTER key causes the data on the screen to be written to the file. The next physical record is entered into the display, and a read is issued for the text portion of the record.

Down Arrow Key

The Down Arrow key causes the next physical record to be displayed. The file is not altered in any manner.

Fl Key

Function key Fl causes the next logical record to be displayed (that is, the record whose record number is currently displayed in the NEXT field). No data is written to the file by this key.

F7 Key

Function key F7 causes the editor to enter the find mode when entered in any field of the record. The message FIND MODE is displayed, and the record displayed is empty. Altering any field causes all records of the JENDAT file to be searched for a record that matches in all of the chosen fields. If no match is found, the editor returns, displaying the NEW RECORD message at the end of the file. This indicates that the search failed. If the record is found, the editor returns to the edit mode, displaying the record that it found.

The text field is searched only when nonblank characters are entered in the text field when the FIND MODE message is displayed. A numeric field is searched when the field is the search. Entering any other valid termination characters returns the editor to edit mode at the displayed record number.

F8 Key

The F8 key returns to the record number field. This permits the user to jump from editing record number 15 to record number 65 by entering F8, followed by 65, and RETURN (SKIP).

TAB Key

The TAB key is acceptable only in the text field. Tab settings are fixed at 1, 8, 13, 31 and 76. Entering TAB causes the read to be reissued at the next tab setting. The fields rotate; that is, a TAB key entered at column 76 returns to column 1. TAB is not accepted in the numeric fields.

Left FIELD Key
The Left FIELD key acts as a reverse tab in the text field. A Left FIELD issued from column 1 is ignored and returns to column 76. In the numeric fields, pressing Left Field causes a return to the previous field, including a return to the text field from the DEF field.

If an error is detected in a numeric field, the message ERROR IN NUMERIC FIELD is displayed. When the error is corrected, the message disappears. Entering CMD corrects the error condition but also returns the user to the main menu. Some errors will not be detected. The RIFLE DECODE procedure terminates the scan when it finds letters while decoding a hexadecimal field. Thus, entering 01WQ in a hexadecimal field places 01 in that field.

16.8.2 PRINT Command.

Entering the PRINT command produces a file for printing. The display is informative only, displaying the record number of the file currently being processed.

16.8.3 QUIT Command.

Entering QUIT closes the JENDAT file and updates record 0 to reflect the current version number and number of records in the file.

16.8.4 REMOVE Command.

Entering REMOVE displays the following message:

```
        REMOVE RECORDS

      FROM ____ TO ____

          ARE YOU SURE? _
```

The numbers entered are validated and may produce either the NUMBER OUT OF RANGE message or the ERROR IN NUMERIC FIELD message. If the response to ARE YOU SURE? is Y, the records from the first number to the second are deleted. This removes the text and enters zero in each numeric field except NEXT, which has the next physical record. To return to the first field from the second, enter the Left FIELD key. This requires that a number for the second field be entered. If the fill character (_) fills the field, the Left FIELD key is not accepted.

16.8.5 SHOW Command.

The SHOW command produces the following prompt:

```
SHOW TEXT FROM RECORD ____
```

The field is validated; and if it is correct, the text of the records beginning with that record and extending through the next 22 records is displayed. If less than 22 records follow the number entered, only those records that exist are displayed.

This command accepts the up arrow, down arrow, F1, and F2 keys in the same fashion as SHOW FILE. F6 is a toggle switch that rolls the file horizontally.

16.8.6 VERSION Command.

The VERSION command produces the following prompt:

```
VERSION NUMBER 01
```

The number shown is the version number checked by SYSJEN to verify compatibility. The number entered replaces the current one. No error checking is done.

16.8.7 MOVE Command.

The MOVE command produces the following prompt:

```
                 MOVE A RECORD

INSERT RECORD ____ BETWEEN ____ AND ____
```

This command causes the first record to be logically inserted between the second and third records. All records which originally pointed to the first record, now point to the logical successor of the first record.

SECTION 17

ANALYZING A SYSTEM CRASH

17.1 OVERVIEW

When DNOS detects a system failure, it displays an error code in the lights of the front panel and idles the CPU. To analyze the problem, copy the memory image to the predefined crash file .S$CRASH on the system disk by pressing HALT and then RUN on the programmer panel. When activity ceases, and the error code is redisplayed, perform an initial program load by pressing HALT, and LOAD. When DNOS is ready, log on to a terminal and study the crash file using the crash analysis utility.

The crash analysis utility can be used to study a crash file or a running system. The paragraphs that follow discuss the commands available with the crash analysis utility and tell when each command is useful. In addition, guidelines are presented for analyzing several particular system crash conditions.

The crash analysis utility is invoked with the XANAL command to SCI. The XANAL command procedure is of the following format:

```
XANAL
  EXECUTE CRASH ANALYSIS UTILITY
        CONTROL ACCESS NAME: pathname@      (ME)
        LISTING ACCESS NAME: pathname@      (ME)
     ANALYZE RUNNING SYSTEM: YES/NO         (NO)
            CRASH FILE NAME: pathname@      (.S$CRASH)
```

The prompts and responses are described in detail below.

CONTROL ACCESS NAME
: This field prompt asks for the access name of the file or device that will be used to issue commands to the utility. Most often, the initial value is accepted, and commands are input from the station at which the XANAL command was issued. In certain cases, you may want to use a standard set of analysis commands from a file. If so, each command must start in column 1 of a separate record of the file.

LISTING ACCESS NAME
: If the crash analysis is to be written to a file, specify a file name. If the station is to receive the listing, accept the initial value.

ANALYZE RUNNING SYSTEM

Accept the initial value of NO if you wish to examine a crash file on disk. Enter a YES to analyze portions of the running system.

CRASH FILE NAME

When examining a running system, accept the initial value for this prompt. When analyzing a crash dump, specify the name of the file. Each time a system crash dump occurs, the information is written to the file .S$CRASH on the system disk. To protect a crash file from being overwritten, copy it to a new file using the CD (Copy Directory) command. Specify .S$CRASH as the input pathname and the desired directory as the output pathname. Upon completion, the crash file will be found as S$CRASH within the directory specified.

When the crash analysis utility is used interactively, the listing comes to the screen at a rapid pace. To stop the display, press the Attention key; to resume the display, press the Attention key again. To exit a command display, press the Command key.

The ANALZ task is an RBID task, therefore you can enter and exit the analysis of a crash to use SCI. Table 17-1 shows the set of commands used with ANALZ to examine a running system or a crash file. The commands are described in detail in the paragraphs which follow.

Table 17-1 Crash Analysis Commands

Command	Display Contents
ALL	Displays from all the commands
AQ	Contents of task active queue
CCB	Channel control blocks
DM	Specific area of memory
FCB	File control blocks
GI	General information
JSB	Job status blocks
LDT	Logical device tables
MM	Memory maps
OVB	Overhead beets
PBM	Partial bit maps
PDT	Physical device tables
PQ	System queues (other than active queue)
QU	No display; terminates session
ROB	Resource ownership blocks
RPB	Resource privilege blocks
SGB	Segment group blocks
SSB	Segment status blocks
ST	Secondary table areas
TA	Task areas in memory
TR	Registers for all tasks
TS	Task status
TSB	Task status blocks
??	List of all available commands

17.2 DETAILS OF CRASH ANALYSIS COMMANDS

When analyzing a crash, the initial step should be to examine the general information about the crash, followed by an examination of task states and then of the detailed information about particular queues and data stuctures. First issue a GI command, then a TS command, and then the relevant structure examination command. All of the commands and their parameters are presented in alphabetic order in the following paragraphs.

ALL

This command lists the information for all the commands available. It is frequently used to process a crash file to a listing file which can then be sent to someone for inspection. When the ALL command is processed, it generates output for the commands in this order: GI, TS, JSB, TSB, SGB, SSB, PDT, FCB, LDT, ROB, CCB, MM, OVB, AQ, PQ, ST, TR, and TA.

AQ

This command causes the display of three lists, the list of tasks on the active queue, the list of tasks waiting for system table area, and the list of tasks on the waiting on memory queue. Each list shows the JSB address, priority, TSB address, and IDs for the tasks involved.

If the queues are empty, the system was idle. If the queues have many entries, the system was busy. Scanning the queues, you can see what tasks were eligible to execute or be loaded into memory. This information is helpful when forcing a crash during a situation where the system appears to be idle or hung in a loop.

CCB

This command shows the channel control blocks for all channels currently in use. It first shows the global channel list from the system table area then the job local list from the JCA of each job in the system. Each job local list is identified by its JSB address.

DM

Before displaying memory, this command solicits data for the following:

LOWER LIMIT - the starting address (rounded to even number)
UPPER LIMIT - the ending address
JSB ADDRESS - a JSB address, an SSB address for any SSB, or 0
TSB ADDRESS - the TSB address for the task memory being displayed
or PDT ADDRESS - the PDT address of a device being viewed
or BEET BIAS - for any beet in memory

If the answer to the JSB ADDRESS prompt is 0, the PDT ADDRESS prompt appears. If the answer to the PDT ADDRESS prompt is 0, the BEET BIAS prompt appears.

The first time DM is used, the prompt for LOWER LIMIT is 0. After the first time, the prompt has an initial value of the previously used LOWER LIMIT. The UPPER LIMIT is initialized with a value >2E greater than the LOWER LIMIT value. The default listing is a three-line display of >30 bytes of data, each line preceded by the address of the first word on that line. The data is shown in hexadecimal and in ASCII equivalent. Initial values for JSB ADDRESS and TSB ADDRESS are the values used previously, after the first DM has been specified.

To examine a physical TILINE address, specify the address as LOWER LIMIT, and specify zero for all other fields of the DM command.

FCB

This command causes a display of all in-memory file structures for files currently in use. It presents the set of structures for each disk device defined for the system, showing for each file the FDB, FCB, and SAT.

GI

General information about a crash is shown by this command, including all of the following for analysis of a crash file. For analysis of a running system, the information beginning with SYSTEM PATCH AREA is presented.

VERSION
The release/version/revision level of the system or crash file being analyzed is shown here. This field provides verification of the current level of software in use.

CRASH CODE
The code is shown as a four-digit hexadecimal value. If the crash code is one of those included in internal tables, an English description of the code is also output. Details on system crash codes are provided in the DNOS Messages and Codes Reference Manual.

EXECUTING TASK
The next item displayed is the TSB address of the executing task at the time of the crash. If the TSB address is shown as 0, there was no executing task at the time of crash. The error was probably within the operating system during a scheduling cycle.

If the crash was an end action crash, this field identifies the task which took end action. If the crash was forced, this field identifies the task which was in a loop when the crash occurred. The information here may not be useful for crashes in the range >13 through >1F (illegal interrupts) or for the >60 series crashes (operating system failures).

EXECUTING TASK JSB
This displays the address of the JSB of the task executing when the crash occurred. Paired with the EXECUTING TASK data, it identifies the task executing at the time of the crash.

LOCATION OF FAILURE
This is the address from which the crash routine NFCRSH was called or, in some cases, the location of an illegal instruction or other cause of the crash.

For crashes in the >60 series, this is not useful. For a >29 crash (unexpected error return), this field identifies whether NFPOPO or NFRTNO encountered the error.

STATUS REGISTER AT TIME OF FAILURE
This shows the contents of the status register when the crash occurred. Bit 8 indicates whether the error occurred while executing in map file 0 (bit 8=0) or in map file 1 (bit 8=1). The last four bits show the interrupt mask. If the interrupt mask is less than <15, the error probably occurred in a DSR or in the clock interrupt handler. This display is not useful for >60 series crashes.

JCASTR
This is the starting address of all JCAs and other table area second segments mapped in by DNOS.

COUNTRY CODE
This entry shows the country code of the system.

IMAGE NAME
The name of the kernel program file that was executing is shown.

MEMORY SIZE
This shows the total memory available to the system while operating. This value is less than or equal to the total physical memory available.

CRASH FILE SIZE
This shows the size of the crash file in use. If this is not as large as MEMORY SIZE, some portions of the crash dump will be missing, and the following message will appear at the start of the GI display:

```
*** WARNING *** ALL MEMORY NOT IN DUMP FILE
```

This generally means the dump is not useful. You should increase the size of your crash file.

CURMAP ADDRESS
This entry shows the address of the current map 0 map file at the time of the crash. Bias values might be used to examine portions of memory. This field is not generally useful if the crash occurred in map file 1.

SYSTEM PATCH AREA
This shows the system patch area in hexadecimal and the ASCII equivalent. The area might be scanned to ensure that all appropriate patches have been applied to the system. The starting address of the patch area should be the same as the symbol NFPATCH in the system link map.

EXECUTING WORKSPACE AT TIME OF DUMP

These registers are the workspace registers of the executing code at the time of the crash. If the crash occurred in map file 0, this area shows the true contents of the registers at the time of the crash. If the crash is for a task taking end action, these registers are those of its end action workspace, which may or may not be those in effect at the time the error occurred.

TOP 64 WORDS OF CURRENT STACK

This display assumes that register 10 of the executing workspace is a stack pointer. The utility displays 32 words preceding the address in register 10 and 32 words following the address in register 10.

This data is most useful if the crash occurred in map file 0, or in a system task.

If the executing code does not use the stack, this area may be useless. Unused memory appears as initialized to >F00D values.

HARDWARE TRAP VECTORS

The hardware interrupt vectors occupy the first 32 words of physical memory and are defined during system generation. These vectors should not be changed unless destroyed by a system task that branches to location 0 by mistake or modifies location 0 when using an incorrectly established address field. If a BLWP instruction is executed to location 0, the return context information of the calling task is stored in locations >1A through >1F.

When the crash code is for an illegal interrupt (>13 through >1F), these vectors are meaningful and should be examined carefully. When a crash code for internal interrupt (>60 through >6F) occurs and the interrupt mask in register 15 of the trap 2 workspace indicates a defined interrupt (mask value minus one), the interrupt trap values should be checked to determine if they are within range. The correct values can be determined by examining locations 0 through >3F of procedure ROOT in the kernel program file.

XOP VECTORS

The XOP vectors occupy the second 32 words of physical memory and are defined during system generation. These vectors should not be changed unless destroyed by a system task in error. Their correct values can be found in the same way as the HARDWARE TRAP VECTORS values.

CLOCK INTERRUPT WORKSPACE

This area shows one of the two clock workspaces. This display workspace shows the current state of the clock interrupt processor. Registers 13 through 15 show the return context of the last entry to the clock interrupt processor. When a crash is forced during a system hang condition, this workspace may point to the location of an infinite loop in a task, that is, the location at which the last clock interrupt occurred.

MACHINE ERROR (TRAP 2) WORKSPACE

This workspace contains diagnostic information about a crash for internal interrupts (>60 through >6F). The context of the crash can be found in registers 13 through 15. If bit 8 of register 15 is set to 1, the error occurred in task driven code (map file 1). If bit 8 is set to 0, the error occurred in system code (map file 0). Register 1 of this workspace carries a status code, reflected as the second digit of the >60 series crash. A >62 crash can be recognized as a forced crash if R14 points to an instruction whose preceding instruction is a zero. If R14 has a value less than that of JCASTR, the error occurred in the root. Otherwise, if in map file 0, check the CURMAP address of the GI information against a system link map for the correct segment of code. If executing in map file 1, consult the appropriate task link map for the correct segment of code.

TRAPPED WORKSPACE

This workspace is found using R13 of the machine error workspace as the starting address.

LOCATIONS AROUND TRAPPED PC

This display shows the 16 bytes before and 16 bytes after the address found in R14 of the machine error workspace.

SVC (XOP 15) WORKSPACE

This workspace is used by both the scheduler and SVC processing. The workspace contains the current state for whichever of those processors last used it. Registers 13 through 15 may contain the return context (workspace pointer, program counter, and status) from the last SVC issued by a task. If the crash occurred in map file 0, this workspace is probably the executing workspace. If the workspace is not that of the scheduler, it may be a DSR workspace; if the starting address is in one of the PDTs, it reflects the workspace of a DSR.

JSB

This command displays the job status blocks for every job in the system. The last JSB presented is that for the system job.

LDT

This command displays, with appropriate headers, all global LDTs, followed by all job-local LDTs identified by JSB address, and finally all task-local LDTs identified by TSB and JSB addresses.

MM

Memory map information, specifying starting address, length, current use, highest address, and free block chain, is presented for each of the following:

* System table area
* Special table areas for segment management and for file management
* Job communication areas for each job in the system

Information is then presented on user memory available, both that available to be swapped and that not available to be swapped. This is followed by tables of linked lists for these structures:

* Free list of user memory
* Deallocate queue
* Time ordered list
* Cache queue
* Write queue
* Buffer table area free list

The MM information is especially useful for analyzing >21 (inconsistent free user memory structure), and >22 (inconsistent table area structures) crashes.

OVB

For each segment in memory, there is an overhead beet (OVB). This command lists all OVBs first by SSB address within the segment manager tables 0 and 1. Several of the initial segments in this list do not have an OVB associated with them (these are segments of resident DNOS). For these segments, the beet preceeding the segment is displayed as the OVB, and will appear as an inconsistent structure. Then the OVBs are listed in sequence in memory order. This list is useful to scan the integrity of structures when a >21 (inconsistent free user memory structure) or >46 (inconsistent segment manager structure) crash occurs.

PBM

This command displays the contents of the partial bit maps for each disk installed in the current system. It is useful when analyzing disk manager crashes.

PDT

The PDT for each device known to the system is shown.

PQ

This command lists the queue server ID and the list of items currently on the queue for each of the following system queues:

* Task bidder
* I/O Utility
* Device I/O Utility
* Disk Manager
* Task diagnostic (kill task) processor
* Job Manager
* IPC task
* Name Manager
* User overlay loader
* Forced roll processor
* Return code processor
* System log formatter
* Accounting log formatter

The PQ lists usually provide clues to any crash. Most of the queues should be empty, except those requiring disk access to handle the queue. If other queues are not empty, the queue entries and queue servers need to be examined for errors. The log queue shows valuable information, too, since it carries the most recent errors sent to the system log.

QU

This command provides no display. It terminates the utility.

ROB

All resource ownership blocks are displayed by JSB identifier.

RPB

This command prompts for an I/O resource pointer. The resource pointer can be an FCB for a file, a PDT for a device, or a CCB for a channel. This command then displays the RPB along with the other relevant structures. For a file, it displays the FDB, FCB, and RPBs. For a device, it displays the PDT and the RPBs. For a channel, it displays the CCB and the RPBs.

SGB

The segment group blocks (SGBs) for all segments currently in memory are displayed, first those in table area 0, then those in table area 1.

SSB

All segment status blocks (SSBs) for all segments currently in memory are displayed, identified by SGB.

ST

The starting address, length, usage, highest address, and contents of the secondary table areas are displayed. This information is provided for each of the special table areas used by segment management and file management and for each JCA in the system.

TA

A memory area is displayed for each of the tasks currently in memory identified by TSB and JSB address.

TR

The registers are displayed for each task in the system, identified by TSB and JSB addresses.

TS

A table is displayed, showing the following information for each task in the system:

* TASK NAME - Installed name of the task
* ID - Installed ID and run-time ID of the task
* WP - Workspace pointer
* PC - Program counter
* ST - Status register
* STATE - Code for the task state (see DSC.TEMPLATE. ATABLE.NFSTAT or the DNOS Messages and Codes Reference Manual for details. The first two digits are the runtime priority of the task, the second two digits are the task state. Tasks in state >04 (terminating) may have an inconsistent display as structures may have been released.
* FLAGS - The first word of task flags from the TSB of the task
* STATION - The station ID from the TSB of the task; if the task was not bid at a station, no station ID is displayed
* TSBADR - The TSB address
* JSBADR - The JSB address
* PROG FILE - The name of the program file from which the task was bid; only the last portion of the pathname is listed

TSB

This command displays each of the TSBs for each of the jobs whose JCA is currently in memory.

??

This command lists all available crash analysis commands.

17.3 GUIDELINES FOR CRASH ANALYSIS

It is impossible to give a set of rules by which crash analysis can be done. There are several general guidelines which should be known, and for specific crashes, there are some specific guidelines. The following paragraphs address general hints first, then some specific suggestions.

In general, an operating system crash occurs when some data structure has been destroyed. The problem in analyzing the crash is then to find what structure has been changed, and what code or combination of circumstances changed the structure. To conduct such an analysis, you must be familiar with the DNOS data structures and be able to detect inconsistencies. Data structure pictures are available in this manual to guide you through the structures displayed by the crash analysis utility. In addition, you will need to understand as much as you can about how the structures are built, used, and released by the relevant DNOS subsystems. The subsystem descriptions in this manual discuss how they use DNOS data structures.

The paragraphs which follow give some specific suggestions for handling particular crash codes.

Codes >13 through >1F

These crashes are for illegal interrupts from devices. Verify that all devices have been specified at the correct interrupt levels during system generation. If this is not the case, perform a new system generation or change the interrupt level using XSCU. If the devices are at the correct interrupt levels, examine the HARDWARE TRAP VECTOR information given by GI and see that it is correct. If not, you need to find the source of the modification. If the information is correct, check the workspace for the interrupt (3 through >F) for clues to the crash.

Code >21 - PMUMGR inconsistent structure

Check register 4 in the MACHINE ERROR (TRAP 2) WORKSPACE data. If it is greater than the value in MEMSIZ, a request has been made to return memory beyond the end of free memory. If the value in Register 4 is less than UADSTR, a request has been made to return memory before the start of free user memory. If neither of these is the case, the free memory list is incorrect. If register 1 is zero, a block of length zero has been specified. If the value at the address in register 10 is greater than MEMSIZ, a block which is too large has been specified. If the value at the address in register 10 is greater than the value in register 11, two blocks of memory overlap. If register 4 is less than register 0, two blocks will overlap when merged in the free list. UADSTR is the start of the free memory list shown by the MM command and MEMSIZ is the end of that list.

Code >22 - NFTMGR inconsistent structure
Examine the executing stack for the return address of the caller of NFTMGR and the address of the item to release in that order.

Code >23 - NFSCHD queueing error
Register 9 of the EXECUTING WORKSPACE has the TSB address of the task that issued the SVC which encountered the error. Verify with the TSB address in EXTSB to see that it is still valid. Check the last SVC issued by that task to find the processor which is in error.

Code >24 - IOBM inconsistent structure
Perform an analysis like that for code >21, using registers 2 and 4. If register 2 has a value less than BTAADD or register 4 has a value greater than UADSTR, a request to return buffer space is incorrect.

Code >26 - PMROLL cannot extend the swap file
This error occurs during task loading; an error indicator is in register 0 of the MACHINE ERROR (TRAP 2) WORKSPACE. If the error indicator is >30, the file is not extendable. If the error is >3F, there is a structure error. If the error is >E0, the disk is full. If the error is >DA, the secondary allocation table is full.

Code >29 - NFPOP or NFMAPO unexpected error returned
Register 11 of the EXECUTING WORKSPACE AT TIME OF DUMP has the address of the return. The return context tells which location was called.

Code >2C - NFENAB - scheduler inhibit count is negative
Register 11 of the MACHINE ERROR (TRAP 2) WORKSPACE points to the caller of NFENAB. That routine may or may not be responsible for the error.

Code >46 - SEGMGR inconsistent structure
The executing stack has the return address, the caller's register 2, and the caller's register 1. Register 1 had the address of the SSB to delete. Use this data to detect an illegal request.

End action codes
Use the TSB list to find the terminating task and examine the diagnostic packet which follows the TSB. The DIA structure identifies the various fields in this packet, showing the error code which caused termination, the

workspace pointer, program counter, and status at the time end action was taken.

Codes >60 through >6F - internal interrupts
Examine the return context (R13 - R15) of the Trap 2 workspace in the GI information for the address of the error. Check the stack for recent routine calls and data saved. If none of the information in the GI information or structures lists is helpful, you might be able to find clues to the crash in the instruction trace kept by the 990/12. The 990/12 hardware keeps 32 words of trace information, reflecting the most recent instruction execution. When an internal interrupt occurs, the interrupt handler copies the hardware trace to the 32 words preceding the scheduler workspace in the system root. The address can be calculated as 32 less than the start of the SVC (XOP 15) WORKSPACE displayed as part of the GI information.

17.4 HARDWARE TRACE INFORMATION

Hardware trace information for a 990/12 can be found in the 64 bytes proceeding the SVC (XOP15) workspace. This data must be used with caution. Because of optimizations being done by the /12, the sequence of hardware instruction pairs may include repetitious data or pairs that appear to logically be out of order.

The format of the trace data is shown in Table 17-2. These words are kept on the processor board for a /12, updated continually with every memory cycle. With a level 2 interrupt, updating ceases after the next memory cycle. The interrupt processor dumps this data to memory proceeding the SVC workspace. The trace data consists of 16 pairs of words of the form shown here as Word 0 and Word 1. The set of entries appears in order of latest instruction executed first, something of a pushdown stack.

Table 17-2 Format of Hardware Trace Information

	Bit	Meaning	Setting
Word 0			
	0	Execution violation	1=Yes
	1	TILINE timeout	1=Yes
	2	Memory data error	1=Yes
	3	Mapping error	1=Yes
	4	Illegal opcode	1=Yes
	5	Privileged instruction attempted	1=Yes
	6	Workspace read/write flag	1=Write
	7	TILINE access flag	1=Access
	8	TILINE R/W FLAG	1=Write
	9	Workspace access flag	1=Access
	10	Instruction fetch flag	1=Fetch
	11-15	Most significant bits of TILINE address	
Word 1			
	0-14	Least significant bits of TILINE address	
	15	Write violation	1=Yes

The overlap of instruction execution in the 990/12 often causes pairs of indicators to be set in a word, showing the activity of the current instruction and the last instruction executed. Some typical examples of word 0 might be

00C0	= Workspace access; TILINE write on previous instruction
0100	= TILINE address
0120	= Instruction fetch
0160	= Instruction fetch; workspace reference on previous instruction
01C0	= TILINE write; TILINE access on previous instruction
02C0	= Workspace write; TILINE write and workspace access on previous instruction

Note that the TILINE address field is not cleared from one trace pair to the next. Therefore, if the TILINE access flag is not set, the TILINE address is meaningless.

Information given about each of the crash codes in the DNOS Messages and Codes Reference Manual might also be helpful in resolving the cause of a system crash. Each of the crash codes is described there, with a general indication of its cause.

SECTION 18

INTERRUPTS AND XOP PROCESSING

18.1 OVERVIEW OF INTERRUPT PROCESSING

The 990 computer has sixteen interrupt levels to serve interrupt requests from various devices and mechanisms. They also have sixteen extended operation codes (XOPs) available to allow extensions to the standard instruction set. Users cannot make use of the XOP feature, since Texas Instruments software makes use of many XOPs and is likely to use the available set for future products.

Interrupt levels are numbered from 0 through 15, with 0 having highest priority. There is a 4-bit interrupt mask which stores the level number of any interrupt presently executing and prevents interrupts of the same or lower levels from interrupting the CPU. The mask is displayed as the last four bits of the status register. The mask may be changed by a LIMI instruction to enable or disable interrupts to the desired level.

Each interrupt level is uniquely associated with a two-word location in memory, referred to as the interrupt trap. This pair of words includes the workspace pointer and program counter values for the program which services the interrupt. The interrupt traps for DNOS are built during system generation and can be found in the sysgen directory in file D$SOURCE (or in its listing form in D$LIST).

DNOS uses the interrupt levels as follows:

* 0 - Power up

* 1 - Power down

* 2 - DNOS internal error

* 5 - CPU clock

* Each of the others is one of: standard device interrupt, expansion chassis interrupt, multiple device interrupt, undefined interrupt

When an interrupt occurs and is not masked out by the current interrupt mask value, CPU control is transferred to the workspace and program counter specified in the trap table. If an undefined interrupt occurs or an internal error is encountered in DNOS, the trap table directs the CPU to a system crash routine. In all other cases, the interrupt trap table causes a transfer to an interrupt processing routine. In DNOS, all interrupt processors are found in the system root.

18.2 OVERVIEW OF XOP PROCESSING

Extended operations (XOPs) are available on the 990 to build the equivalent of machine instructions not provided in the hardware. There are 16 extended operation codes (levels) available. One of these, level 15, is reserved by DNOS for handling supervisor calls (SVCs); levels 9 through 12 are used by the DNOS performance package, and the remaining levels are reserved for future DNOS use.

When the CPU encounters an XOP instruction, it tests first to determine if a hardware XOP processor is present. If so, control is passed to that processor for execution. If no hardware processor is present, control is transferred to a software routine via a table of processor addresses built during system generation. This table is referred to as the XOP transfer table and can be found in the sysgen directory in the file D$SOURCE (or in its listing form in D$LIST).

18.3 BUILDING AN XOP PROCESSOR

In special situations, a programmer may wish to implement an instruction or devise a service of DNOS which is not available in the supplied version of DNOS. To meet these situations, DNOS allows the programmer to build his own supervisor calls (SVCs), as described in the DNOS Systems Programmer's Guide. In cases where this feature is not sufficient, DNOS system software may need to use an XOP processor.

To add an XOP processor to DNOS, a processor must be written and details must be provided during system generation.

18.3.1 System Generation Requirements for User XOPs.

The XOP related prompts during system generation are described in Table 18-1. The system generation program inserts the XOP processor entry point and workspace address into the XOP transfer table it builds for the system. The system generation program also includes the XOP processor object module in the linkstream for DNOS.

Table 18-1 System Generation Prompts for XOPs

Prompt	Response required
ENTITY?	XOP
XOP LEVEL?	Level number (0 through decimal 14) that is to be used
PC label?	Entry point label of the XOP processor
WP label?	Workspace label of the XOP processor
Pathname?	Name of the file that contains the object module for the XOP processor

18.3.2 XOP Processor Details.

When an XOP instruction is executed, control transfers to the XOP processor via the XOP transfer table. In the workspace of the XOP processor, the following registers are loaded and must not be destroyed by the processor:

- Register 11 - the address of the XOP operand, relative to the calling task address space

- Register 13 - the requesting task workspace pointer

- Register 14 - the requesting task program counter

- Register 15 - the requesting task status register

The XOP processor should execute quickly and be relatively short. It cannot issue supervisor calls, but must perform all operations locally. It can access the calling task address space by using long distance instructions with the saved map file of the calling task. This map file is at the offset TSBML1 in the task status block (TSB) pointed to by the system pointer named EXTSB (executing task TSB). Figure 18-1 shows long distance access to three parameters from the calling task address space.

The XOP processor must define (DEF) its entry point and its workspace address, and it must reference (REF) the system return point, NFTRTN. If it makes use of EXTSB, it must copy two system templates, using the following statements:

```
COPY DSC.TEMPLATE.ATABLE.TSB
COPY DSC.TEMPLATE.COMMON.NFPTR
```

The first statement copies a template which has the offset TSBML1 within a TSB. The second copies a set of system pointers that includes EXTSB. Other data structures might also be accessed by an XOP processor. Consult the section on data structure pictures in this manual for details on the structures to be used.

```
*------
*  THIS EXAMPLE SHOWS HOW AN XOP PROCESSOR
*  INTERFACES WITH THE OPERATING SYSTEM TO ACCESS A CALLER'S
*  TASK AREA AND RETURN TO THE CALLER.  THE EXAMPLE IS FOR
*  XOP LEVEL 10, WHERE THE PROCESSOR MOVES N WORDS OF DATA.
*
*  CALLED BY:     XOP   @ARGS,10
*       WITH:
*
*         ARGS     DATA X,Y,N   WITH X = SOURCE ADDRESS
*                                    Y = DESTINATION ADDRESS
*                                    N = NUMBER OF WORDS TO MOVE
*------
* EQUATES
*
       COPY DSC.TEMPLATE.ATABLE.TSB        TO ACCESS TSB FIELDS
* GLOBAL DATA
*
       COPY DSC.TEMPLATE.COMMON.NFPTR      TO ACCESS EXTSB
*
       REF  NFTRTN        SYSTEM RETURN POINT
       IDT  'USRXOP'
       DEF  USRXOP        ENTRY POINT FOR SYSGEN RESOLUTION
       DEF  WPAD          WORKSPACE ADDRESS FOR SYSGEN
WPAD   BSS  32            LOCAL WORKSPACE
USRXOP EQU  $             ROUTINE STARTS HERE
       MOV  @EXTSB,R8     GET CALLER TSB ADDRESS
       AI   R8,TSBML1     POINT TO THE MAP FILE
       LDS  *R8           USING XOP OPERAND IN R11
       MOV  *R11+,R1      GET ADDRESS OF ARGUMENT X
       LDS  *R8
       MOV  *R11+,R2      AND ADDRESS OF Y
       LDS  *R8
       MOV  *R11,R3       AND OF N
LOOP   EQU  $             MOVE N WORDS; ASSUME GOOD ADDRESSES
       LDS  *R8           GET A WORD OF SOURCE INFORMATION
       MOV  *R1+,R4
       LDD  *R8           MOVE IT TO DESTINATION
       MOV  R4,*R2+
       DEC  R3            COUNT DOWN
       JNE  LOOP          MORE TO DO IF COUNT POSITIVE
       B    @NFTRTN       ELSE RETURN TO OPERATING SYSTEM
       END
```

Figure 18-1 XOP Processor

SECTION 19

SPECIAL SVCs

19.1 OVERVIEW

This set of SVC operations is supported for DNOS operating system tasks only. They are not documented in user manuals and must not be issued by user-written code.

Each of the SVC blocks is shown with a hexadecimal offset at the left of each word. The upper right of the block shows special conditions which must be met, such as alignment on a word boundary, if such a condition is relevant.

19.2 I/O SVCs

Several of the subopcodes of the I/O SVC can be issued only by operating system tasks. They generate error conditions otherwise.

19.2.1 DSRTPD Diagnostics Control (Subopcode >08).

DSRTPD supports diagnostics control of the communications hardware. The mechanism to support this is I/O subopcode >08. The extended call block is used for further subopcodes and parameters.

The following functions are supported:

SUBOPCODE 08 FUNCTIONS

SUB-SUBOPCODE	DESCRIPTION
56	Write interface image
66	Read interface image

The call block has the following format:

```
     Hex Offset                    Align on Word Boundary
                +------------------+-----------------+
        >00     |        00        |  <Return Code>  |
                +------------------+-----------------+
        >02     |        08        |     LUNO        |
                +------------------+-----------------+
        >04     |  <System Flags>  |  User Flags     |
                +------------------+-----------------+
        >06     |               Unused               |
                +------------------+-----------------+
        >08     |               Unused               |
                +------------------+-----------------+
        >0A     |               Unused               |
                +------------------+-----------------+
        >0C     |               Unused               |
                +------------------+-----------------+
        >0E     |               Unused               |
                +------------------+-----------------+
        >10     |             Parameters             |
                +------------------+-----------------+
        >12     |             Parameters             |
                +------------------+-----------------+
        >14     |  Subopcode       |   Unused        |
                +------------------------------------+
```

Byte 5, Bit 6 = 1 (Extension flag)

Byte 14, Bit 0 = Error bit

19.2.1.1 Write Interface Image.

Sub-subopcode >56 performs the diagnostic write. The call block format depends on the kind of interface card that is in use at the port. For ports using the COMM INTERFACE board, the format is as follows:

```
        Byte             Description

         10              Bit  0 - Character length select 1
                              1 - Character length select 0
                              2 - Sync mode selection
                              3 - Odd parity select
                              4 - Alternate clock select
                              5 - Clock select A2
                              6 - Clock select A1
                              7 - Clock select A0
         11              Modem leads (0=low, 1=high)
                         Bit  0 - Self test mode
                              1 - Transmit break
                              2 - 1 = 2 stop bits, 0 = 1 stop bit
```

	3 - Echo enable
	4 - Parity enable
	5 - Receiver enable
	6 - Request to Send
	7 - Data Terminal Ready
	Interface Control
12	Bits 0-7 - Sync character-first load
	DLE character - second load
13	Bit 0 - Analog loopback
	1 - Half duplex
	2 - Master reset
	3 - Pulsed modem lead out
	4 - Reserved modem lead out
	5 - Secondary request to send
	6 - Clock select B1
	7 - Clock select B0
14	Subopcode (>56)
15	Reserved for board compatibility

For the TTY/EIA card the format is as follows:

Byte	Description
10	Modem leads
	Bit 0 - Ignored
	1 - Data terminal ready
	2 - Request to send
	3 - Clear read request
	4 - Clear write request
	5 - Clear new status
	6 - Enable interrupts
	7 - Diagnostic mode
11	Ignored
12	Ignored
13	Ignored
14	Subopcode (>56)
15	Reserved for board compatibility

19.2.1.2 Read Interface Image.

Sub-subopcode >66 performs the diagnostic read. Modem and interface information is returned in the call block as follows:

For the COMM board:

Byte	Description
10	Interface information
	Bit 0 - Write request
	1 - Interrupt summary

	2 - Timer expiration 3 - New status flag 4 - Scan Busy 5 - Transmit underrun 6 - Readable copy of sync selection 7 - Read request
11	Modem leads Bit 0-1 Unused 2 - Data carrier detect 3 - Ring indicator 4 - Reserved modem lead out 5 - Secondary request to send 6 - Clear to send 7 - Data set ready
12	Bits 0-3 - Unused 4 - Parity error 5 - Framing error 6 - Receiver overrun 7 - Receive error summary
13	Bits 0-7 - Receive data byte
14	Subopcode
15	Reserved for board compatibility

For the TTY/EIA board:

Byte	Description
10	Bits 0-7 - Receive data byte
11	Modem leads Bit 0 - Interrupt 1 - Data set ready 2 - Data carrier detect 3 - Read request 4 - Write request 5 - Ring indicator (cable 2265151-0001) reverse channel receive (other cables) 6 - Timing error (overrun) 7 - Xmit in progress
12	Ignored
13	Ignored
14	Subopcode
15	Reserved for board compatibility

19.2.2 Communications DSR Diagnostics Control (Subopcode >08).

The DSRs used by communications software use a call block like DSRTPD to support diagnostic control of the communications hardware. The following functions are supported, using the specified subopcodes in the extended call block at offset >14. Note that for each subopcode that bits 0 through 3 are defined specially for that operation.

Sub-Subopcode	Meaning
>X0	Abort/Timeout
>X1	Open
>X2	Close
>X3	Write
>X4	Read
>X5	Chained write
>X6	Miscellaneous channel commands - diagnostics
>X7	Reserved for Protocol - immediate
>X8	Reserved for Protocol - data
>X9	Reserved for Protocol - data
>XA	Reserved for Protocol - data
>XB	Reserved for Protocol - data
>XC	Miscellaneous board - data
>XD	Miscellaneous board - immediate
>XE	Stand-alone diagnostics (not supported)
>XF	Immediate diagnostics

19.2.3 Open Unblocked (Subopcode >13).

The Open Unblocked SVC block has the following format:

```
                                    Align on word boundary
 Hex Offset   *----------------+-----------------*
    >00       |       00       |  <Return Code>  |
              +----------------+-----------------+
    >02       |       13       |      LUNO       |
              +----------------+-----------------+
    >04       ~             Reserved             ~
              ~                                  ~
              *----------------------------------*
          (>0A - maximum size)
```

It is used by I/O utility and several other utilities to access files in a special way. This same opcode is used by Unload Volume to dump statistics to the system log. In this case, the LUNO field is ignored and bytes 6 and 7 point to a special area designating use of VCATALOG.

19.2.4 Close Without Updating FDR (Subopcode >14).

The Close Without Updating FDR is used by the directory utilities in conjunction with the Open Unblocked operation when copying a file. Since the normal copy of a file description record (FDR) would change the date of latest modification, it cannot be used by the directory utilities. The format of the block is as follows:

```
                            Align on word boundary
Hex Offset    *----------------+-----------------*
   >00        |       00       |  <Return Code>  |
              +----------------+-----------------+
   >02        |       14       |      LUNO       |
              +----------------+-----------------+
   >04        ~             Reserved             ~
              ~                                  ~
              *----------------------------------*
        (>0A - maximum size)
```

The Open Unblocked SVC is used by the directory utilities to allow reading of any file as a relative record file.

19.2.5 DSRTPD Communications Control - (Subopcode >15).

DSRTPD supports task access to device dependent communications control using subopcode >15. The call block is the same format as the Write ASCII subopcode. Further subopcodes and parameters are contained inside the data buffer. Most of these functions are also performed by the SCI command MHPC.

```
Hex Offset                        Align on Word Boundary
            +---------------------+-------------------+
   >00      |          00         |  <Return Code>    |
            +---------------------+-------------------+
   >02      |          15         |     LUNO          |
            +---------------------+-------------------+
   >04      |   <System Flags>    |  User Flags       |
            +---------------------+-------------------+
   >06      |Buffer Address, Secondary Control Block  |
            +---------------------+-------------------+
   >08      |              Unused (0)                 |
            +---------------------+-------------------+
   >0A      |           Buffer Byte Count             |
            +-----------------------------------------+
```

Data Buffer Descriptions

Byte	Value
0	Subopcode
1	Reserved (>00)
2-N	Parameters if needed

The following functions are supported; with the subopcodes indicated being placed in the data buffer.

Opcode 15 Functions

Subopcode	Function
>16	Modify timing characteristics
>17	Modify line characteristics
>18	Modify terminal type
>19	Modify special characters
>1A	Connect
>1B	Flush character queue
>1C	Set file transfer parameters
>1D	Set exclusive access
>1E	Set shared access

19.2.5.1 Set File Transfer Parameters >1C.

This command enables selection of a parity checking mode, selects timeouts, selects a parity error substitute character, and disables the DC3-driven functions: bid, hold output, abort task, and timeout. Parameters are located as follows in the call block:

Btye		Meaning
2-3		Primary timeout, read direct
4-5		Secondary timeout, read direct
6		Parity error substitute character
7,	Bit 0	Suppress echo=1, echo=0
	Bit 1	Unused
	Bit 2	Enable transmit parity=1
	Bits 3-4	Transmit parity type 00=even 01=odd 10=mark 11=space
	Bit 5	Enable receive parity=1
	Bits 6-7	Receive parity type (Same as transmit)

The values so selected disappear when the terminal is disconnected.

19.2.5.2 Modify Timing Characteristics >16.

The default timeouts are changed. Values are gathered from the primary control block:

Byte	Value (250 ms increments)
2-3	Read timeout
4-5	Write timeout
6-7	Read direct timeout (first character)
8-9	Read direct timeout (other characters)

19.2.5.3 Modify Line Characteristics >17.

This call modifies the line configuration with the following options:

Byte		Value
2		LTA character (00=don't change)
3		Speed **
4,	Bit 0	Half-duplex = 1
	Bit 1	Switched
	Bit 2	Disabled
	Bit 3	Auto-disconnect enabled
	Bit 4	Require DLE+EOT for auto-disconnect
	Bit 5	SCF ready/busy monitor
	Bit 6	Exclusive access
	Bit 7	LTA enable (half-duplex only)

**The following table gives the speed translations for the value of byte 20

Value	Speed (ASYNC BPS)
0	110
1	300
2	600
3	1200
4	2400
5	4800
6	9600
-1	300 or 1200 depending on state of pin 12 at the COMM I/F. This is used for automatic speed selection in conjunction with VA3400 and 212A modems.

19.2.5.4 Modify Terminal Type >18.

This call allows parameters related to the expected terminal type to be altered.

Byte		Value
2		Terminal model** (3=703, 7F=763)
3,	BIT 0	Echo = 0

No echo = 1

**The following table gives the terminal type translation:

Value	Terminal Type
03	703
07	707
2B	743
2D	745
3F	763
41	765
51	781
53	783
55	785
57	787
78	820
7D	825

19.2.5.5 Modify Special Characters >19.

This call modifies the characters used for end of record and end of file.

Byte	Value
2	End of record (00=don't change)
3	End of file (00=don't change)

19.2.5.6 Connect >1A.

This call establishes a connection in the indicated way. If bit 0 of the user flags is set (INITIATE I/O) the task is not suspended pending the establishment of connection. revisifon bar on If the TPD is not the call originator, DTR is asserted only after Ring Indicator or Data Set Ready is detected. Once Ring or Data Set Ready is detected, the timeout reverts to 10 seconds for the completion of the connection. Thus, if a port is set to answer incoming calls with an infinite time-out, and some non-modem device calls in, the DSR will timeout the call 10 seconds after the phone rings. In full-duplex environments, Data Carrier Detect must be sensed for the call to complete successfully.

Byte	Value
2	Assert RTS (00= do not assert)
3	Assert DTR (00= do not assert)
4,5	Timeout (250 ms increments, 0=infinite)

19.2.5.7 Flush Character Queue >1B.

This call removes any characters buffered in the character queue of the KSB. If an extended call block is used with bit 4 of extended user flags set, the DSR is placed in 8-bit data mode. If an extended call block is not used or if bit 4 is not set, the DSR is placed in normal mode.

19.2.5.8 Set Exclusive Access >1D.

This call places the port under control of file transfer tasks. These tasks have bit 5 of the user flags in the PRB set to one on opens.

19.2.5.9 Set Shared Access >1E.

This call releases the port to tasks that do not have bit 5 of the user flags set to one on opens.

19.2.6 VDT Extended Edit Flags (Subopcode >15).

Device dependent edit modes for the 911, 931 and 940 VDTs are accessed like the DSRTPD Communications Control (Subopcode >15). When using subopcode 00 in the data buffer, bytes 2 through 5 of the data buffer form 2 words of flags. The following bit setting cause the described functions to be performed.

First flag word (Bytes 2-5 of the subopcode 15 data buffer)

Bit 0 - 931,940 - enable pass through mode

1 - 931,940 - in pass through mode, terminate read on EXT (>03)

2 - 931,940 - in pass through mode, terminate read on ESC-) pair

3 - 931,940 - allow extended event characters
911 - map hardware generated codes 00 ->1F to event characters in range >E0 - >FF

4 - reserved

5 - 931,940 - allow ESC and SOH characters in Write ASCII BUFFER (access to reverse video, underline, and blink)

6 - reserved

7 - reserved

8 - 911,915 - report modified data to caller on Read ASCII (DSR sets bit 7 of system flags byte)

9 - 911,915 - extended character validation (invalid characters are not echoed, error flag not set, beep occurs if warning beep flag is set)

10 - 911,915 - Suppress null characters on input, allow null character on either 7 or 8 bit Write

ASCII
11 - 911,915 - convert embedded nulls to spaces on Read
ASCII
12 - Kanji - toggle screen edit mode and 911 emulation
13-15 - reserved - must be set to zero

Second flag word (bit set to 1 indicates key is enabled for 911 and 940 as an event key in the PDT. The existence of the second flag word enables its use. (All the indicated keys are mapped to the 913 code, as they would be in the WP mode that is no longer available.

0	Erase Field
1	Right Field
2	Left Arrow at left margin
3	Tab
4	Down Arrow
5	Skip
6	Home
7	Return
8	Erase Input
9	Blank Gray (default anyway)
10	Delete Character
11	Insert Character
12	Right Arrow at right margin
13	Enter
14	Left Field
15	Up Arrow (default)

The first three bits of the first flag word allow the "pure pass-thru" support needed to use the 940 in block mode or as a replacement for the old SVCs >8 and >18 character mode. These functions are not perceived to be particularly useful, but will be left in the DSR as a hook to any features not supported by our software. An application that uses these functions must restore the screen image and terminal state to standard modes on exit from pass thru mode.

Bit 3 (the fourth bit) of the first flag word allows the 3270 package (and any others) to get at the extended function keys on the 940 keyboard.

Bit 5 allows access to setting the extended display attributes of reverse video, underline and blinking.

19.2.7 Asynchronous Multiplexor Operation (Subopcode >15).

Two requests for the Modify Device Characteristics SVC (opcode >00, subopcode >15) are supported for both VDT and printer DSRs that execute on buffered TILINE multiplexers (CI403/CI404). These two requests are Read UART Registers and Write UART

Registers. These requests are provided for the use of diagnostic programs. Their use requires detailed knowledge of the CI403/CI404 controllers and the WD8250 UART used on the controllers. Refer to the CI403/CI404 hardware documentation for more detailed information.

The Read and Write UART Registers both directly access the functions of slave word 1 of the CI403/CI404 TILINE Peripheral Control Space (TPCS). A diagnostic task is allowed direct access to seven UART registers for each channel of the multiplexer.

19.2.7.1 Write UART Registers.

The primary function of the Write UART Registers request is setting and resetting specific UART inputs and RS-232-C signals for a diagnostic program. The request is implemented by using the Modify Device Characteristics SVC (I/O subopcode >15) with the sub-subopcode >31. This opcode provides direct, device-dependent access to slave word 1 of the CI403/CI404 TPCS. The operation must be issued with an extended call block; otherwise, an error is returned.

Parameters furnished by the online diagnostic task include the output data buffer length, the contents of the data buffer, the UART register number to which the data is to be written (call block byte 18), and the register data (call block byte 19). If the device (PDT) to which the operation is issued is not in diagnostic mode, the request is rejected. Figure 19-0 shows the call block formats and applicable fields in the Write UART Registers request.

```
     +---------------------------------+
 0   |      >00       |   STATUS BYTE  |
     |----------------+----------------|
 2   | I/O SUB-OPCODE |      LUNO      |
     |----------------+----------------|
 4   |  SYSTEM FLAGS  |   USER FLAGS   |
     |----------------+----------------|
 6   |       DATA BUFFER ADDRESS       |-----------+
     |---------------------------------|           |
     |            RESERVED             |           |
 8   |      LOGICAL RECORD LENGTH      |           |
     |---------------------------------|           |
10   |         CHARACTER COUNT         |           |
     |---------------------------------|           |
12   | REPLY BLOCK ADDRESS - RESERVED  |           |
     |---------------------------------|           |
14   |       EXTENDED USER FLAGS       |           |
     |---------------------------------|           |
     |             RESERVED            |           |
16   | FILL CHARACTER |  EVENT BYTE    |           |
     |---------------------------------|           |
18   | REG #  |  RES  | REGISTER DATA  |           |
     |---------------------------------|           |
     |             RESERVED            |           |
20   | FIELD ROW POS. | FIELD COL POS. |           |
     +---------------------------------+           |
                                                   |
     +---------------------------------+           |
     |       >31      |     IGNORED    |<----------+
     +---------------------------------+
```

Figure 19-1 Write UART Register Format

Affected fields of the SVC call block and their meanings are as follows:

Byte	Bit	Meaning
0		Specifies SVC call type. Enter >00 for I/O.
1		Used to return operation error codes.
2		Subopcode. Enter >15 for Modify Device Characteristics
3		LUNO. Use the LUNO from the diagnostic assign.
4		System flags. Standard OS definitions apply.
5		User flags. Standard OS definitions apply.
	5	Extended call block. 1 = Read or Write UART Registers.
6,7		Data buffer address. Points to the buffer that contains the sub-subopcode.

8,9		Logical record length. Not used.
10,11		Character count. 2 = Read or Write UART Registers.
18	7-4	Register number (0-7) left justified in byte. Specifies the UART register to which the operation is directed.
19		Register data. For the Write UART subopcode, specifies data to be written.

19.2.7.2 Read UART Registers.

The primary function of the Read UART Registers request is sensing UART inputs and RS-232-C signals for a diagnostic program. The request is implemented by using the Modify Device Characteristics SVC (I/O subopcode >15) with the sub-subopcode >32. This opcode provides direct, device-dependent access to slave word 1 of the CI403/CI404 TPCS. The operation is issued with an extended SVC call block.

Parameters furnished by the online diagnostic task include the output data buffer length, the contents of the data buffer, and the UART register number from which the data is to be read (call block byte 18). The register data is returned in call block byte 19. If the device (PDT) to which the operation is issued is not in diagnostic mode, the request is rejected. Figure 19-2 shows the SVC call block formats and applicable fields in the Read UART Registers request.

```
      +---------------------------------+
  0   |       >00       |  STATUS BYTE  |
      |-----------------+---------------|
  2   | I/O SUB-OPCODE  |     LUNO      |
      |-----------------+---------------|
  4   |  SYSTEM FLAGS   |  USER FLAGS   |
      |-----------------+---------------|
  6   |       DATA BUFFER ADDRESS       |-----------+
      |---------------------------------|           |
  8   |LOGICAL RECORD LENGTH - RESERVED |           |
      |---------------------------------|           |
  0   |         CHARACTER COUNT         |           |
      |---------------------------------|           |
 12   |       REPLY BLOCK ADDRESS       |           |
      |---------------------------------|           |
 14   |       EXTENDED USER FLAGS       |           |
      |---------------------------------|           |
      |            RESERVED             |           |
 16   | FILL CHARACTER |  EVENT BYTE    |           |
      |---------------------------------|           |
 18   | REG #  |  RES  | REGISTER DATA  |           |
      |---------------------------------|           |
      |           RESERVED              |           |
 20   | FIELD ROW POS. | FIELD COL POS. |           |
      +---------------------------------+           |
                                                    |
      +---------------------------------+           |
      |       >32       |    IGNORED    |<----------+
      +---------------------------------+
```

Figure 19-2 Read UART Registers Format

Affected fields of the SVC call block and their meanings are as follows:

Byte	Bit	Meaning
0		Specifies SVC call type. Enter >00 for I/O.
1		Used to return operation error codes.
2		Subopcode. Enter >15 for Modify Device Characteristics.
3		LUNO. Use the LUNO from the diagnostic assign.
4		System flags. Standard OS definitions apply.
5		User flags. Standard OS definitions apply.
	5	Extended call block. 1 = Read or Write UART Registers.
6		Data buffer address. Points to the buffer that contains the sub-subopcode.
8		Logical record length. Not used.

10,11		Character count. 2 = Read or Write UART Registers.
18	7-4	Register number (0-7) left justified in byte. Specifies the UART register to which the operation is directed.
19		Register data. For the Read UART subopcode, the data read from the device is returned in this location.

19.2.8 TILINE Diagnostic Port (Subopcode >16).

The TILINE Diagnostic Port operation allows the passing of a sixteen byte controller image buffer to a device from a nonprivileged task. The TILINE controller image after the execution of the command is returned in the controller image buffer. This subopcode is only valid for disk and magnetic tape devices and is used by online diagnostics.

The call block has the following format:

```
Hex Offset                  Align on Word Boundary
               +-----------------+-----------------+
   >00         |        0        | <Return Code>   |
               +-----------------+-----------------+
   >02         |       16        |      Luno       |
               +-----------------+-----------------+
   >04         | <System Flags>  |   User Flags    |
               +-----------------------------------+
   >06         |  TILINE Image Buffer Address      |
               +-----------------------------------+
   >08         |         Reserved = >10            |
               +-----------------------------------+
   >0A         |         Reserved = >10            |
               +-----------------+-----------------+
   >0C         |Binary OS Ver/Rel| Diagnostic Flags|
               +-----------------+-----------------+
   >0E         |         Dynamic Passcode          |
               +-----------------------------------+
         (>10 - maximum size)
```

Byte	Description
3	LUNO assigned to the disk or mag tape
4	Flags set by system - when set mean:

Bit 0	-	LUNO is busy
Bit 1	-	Error
Bit 2-7	-	Reserved (set to 0)

5	Flags set by user - when set mean:

Bit 0	-	Initiate I/O
Bit 1-6	-	Reserved (set to 0)
Bit 7	-	No retries are to be performed

6-7 Address of TILINE image buffer. This buffer will contain the TILINE controller image after the command completion. Must begin on word boundary and be sixteen bytes in length.

12 This byte must contain the version and release of the operating system in binary. For example, to execute this I/O supervisor call on DNOS 1.1, this byte must contain the value >11.

13 Diagnostic flags set by the user - when set mean:

Bit 0 - 0 = Bytes 10 and 11 in the TILINE image buffer do not contain a logical address,
1 = Bytes 10 and 11 in the TILINE image buffer contain a logical address

Bit 1-7 - Reserved (set to 0)

14-15 Dynamic passcode - must contain the current value of the system minute.

When bit 0 of the "diagnostic flags" (byte 13) is set, the controller image buffer is modified by the operating system prior to execution of the command. In particular, bytes 10 and 11 are assumed to contain a logical address; this address is converted to a physical 21 bit TILINE address which is inserted into bytes 10/11 (LSB), and bits 4-7 of byte 13 (MSB) of the TILINE image buffer.

CAUTION

> If the command passed in the TILINE controller image transfers data to or from the device, it is the responsibility of the task issuing the Diagnostic Port operation to set bit 0 of the diagnostic flags to 1, and provide the logical address (bytes 10 and 11) and byte length (bytes 8 and 9) of the read/write buffer in the controller image buffer.

The operating system performs address space verification when bit 0 of the diagnostic flags is set. If this bit is not set, no address space verification is performed. The address space verification checks that the buffer address (bytes 10 and 11) and buffer byte length (bytes 8 and 9) fit entirely in one segment of the issuing task. Write and execute protection of the segment are not checked.

The unit select field is ignored in the TILINE controller image buffer; the DSR sets the unit select field properly to indicate the device to which the luno is assigned. It should be noted that certain fields in the TILINE controller image buffer have meaning only after the command is executed. The controller image buffer has the following format:

FOR DISK DEVICES:

```
     +-------------------------------------------+
 0   |               DISK STATUS                 |
     +-------------------------------------------+
 2   |                 COMMAND                   |
     +-------------------------------------------+
 4   |              FORMAT/SECTOR                |
     +-------------------------------------------+
 6   |                CYLINDER                   |
     +-------------------------------------------+
 8   |                  COUNT                    |
     +-------------------------------------------+
10   |         LOGICAL ADDRESS (16 BIT)          |
     +-------------------------------------------+
12   |           SELECT/MSB ADDRESS              |
     +-------------------------------------------+
14   |           CONTROLLER STATUS               |
     +-------------------------------------------+
```

FOR TAPE DEVICES:

```
      +--------------------------------------------+
 0    |          TAPE TRANSPORT STATUS             |
      +--------------------------------------------+
 2    |        READ OVERFLOW STATUS COUNT          |
      +--------------------------------------------+
 4    |        READ OVERFLOW STATUS COUNT          |
      +--------------------------------------------+
 6    |              READ OFFSET                   |
      +--------------------------------------------+
 8    |                 COUNT                      |
      +--------------------------------------------+
10    |        LOGICAL ADDRESS (16 BIT)            |
      +--------------------------------------------+
12    |       COMMAND/SELECT/MSB ADDRESS           |
      +--------------------------------------------+
14    |            STATUS/CONTROL                  |
      +--------------------------------------------+
```

The following error codes are unique to the Diagnostic Port. All other error codes that occur when using the Diagnostic Port are standard SVC error codes.

Error	Meaning
>00E8	Invalid TILINE Diagnostic Port passcode. The passcode is invalid if the following conditions are not met in the SVC call block: Byte 8 = >00, Byte 9 = >10, Byte 10 = >00, Byte 11 = >10, Byte 12 = Binary OS version/release (described above), Byte 14-15 = value of system minute.
>00E9	Invalid TILINE command used with TILINE Diagnostic Port.

The disk DSR handles TILINE Diagnostic Port requests in a unique manner. (Other than returning the TILINE controller image, the tape DSR does not handle Diagnostic Port requests in a unique manner.) The disk DSR insures that no requests are outstanding on any of the devices attached to the same controller (as the device receiving the request), before issuing the Diagnostic Port operation. After all outstanding requests are completed, the Diagnostic Port operation will be initiated. Following the completion of the Diagnostic Port operation, all requests queued to the devices attached to the same controller are initiated. During the interval between the receipt of a Diagnostic Port request and the completion of that request, no other requests will be initiated to any device attached to the same controller.

CAUTION

> Because the disk DSR was not designed to handle seek and restore commands initiated by the user, a seek or restore command should never be issued in the TILINE image buffer. System error conditions will result if a seek or restore is issued.

19.2.9 Read with Initial Value (Subopcode >17).

The Read with Initial Value subopcode is implemented for the 911, 931 and 940 devices. This operation performs the same functions as Read ASCII, except that the field initial value is taken from the user's buffer rather than from the terminal display memory. When using this operation, the user must rationalize any discrepencies between the visible initial value in the field and that which is passed to the DSR in the buffer. The operation is used by 3270 communications software.

The following call block is used:

Align on Word Boundary

Dec	Hex		
0	0	00	<Return Code>
2	2	17	LUNO
4	4	<System Flags>	User Flags
6	6	Data Buffer Address	
8	8	Read Character Count	
10	A	<Actual Read Count>	
12	C	Validation Table Address	
14	E	Extended User Flags	
16	10	Fill Character	<Event/Byte>
18	12	Cursor Position Row	Column
20	14	Field Beginning Row	Column

(>16 - Maximum size)

Byte	
04	System flags

0 - Busy
1 - Error
2 - EOF
3 - Event
7 - Modify data tag (operator pressed a valid data key or erase key)

05 User flags
5 - Extended block - must be set 1

08-09 Output character count on initial read option - size of initial value in the buffer

0A-0B Actual count of characters read (always less than or equal to size of initial value)

0C-0D Specify the address of a validation table if character validation is set in the extended user flags. Otherwise set to zero.

0E Extended user flags
0 - Field start position
1 - Intensity
2 - Blink cursor
3 - Graphics
4 - Eight-bit characters (intensity bit)
5 - Task edit
6 - Beep
7 - Right boundary

0F Extended user flags
0 - Cursor position in read field
1 - Fill character
2 - Do not initialize field
3 - Require termination char for return
4 - No echo
5 - Character validation
6 - Ignored
7 - Warning beep

11 Programmable key or blank returned

The "do not initialize field" flag set to 1 indicates that the DSR will not replace the data on the screen by what is in the buffer before doing the Read operation.

19.2.10 Assign Diagnostic Device (Subopcode >94).

The Assign Diagnostic Device is issued by a task that is assigning a LUNO to a device that is in the diagnostic state. The call block is of exactly the same format as the Assign LUNO (subopcode >91); but subopcode >94 is required to assign a LUNO to a device that is in the diagnostic state. Only one task can successfully execute this SVC at any one time. Other tasks attempting the assign will receive a >9C error.

19.2.11 Attach File (Subopcode >A0).

The Attach File SVC is used to reserve access to a file. The FCB representing the file to be attached is built (or located if already in memory). The SVC is currently used by the O.S. to support job local temporary files.

If the request is the first attachment to this file by this job, an ROB is built to point to the FCB representing the file. An attachment number between 0 and 255 (unique to this job) is generated and placed in the ROB. (See the discussion of Detach File by Number SVC for details of it's use.) The LUNO count for the FCB is incremented.

If the File was already attached by this job, the count of attach operations in the ROB is incremented.

The Attach Resource SVC block has the following format:

```
                                  Align on word boundary
 Hex Offset  *----------------+-----------------*
    >00      |       00       |  <Return Code>  |
             +----------------+-----------------+
    >02      |       A0       |    Reserved     |
             +----------------+-----------------+
    >04      ~                                  ~
             ~                                  ~
    >06      ~             Reserved             ~
             ~                                  ~
             +----------------------------------+
    >16      |         Pathname Pointer         |
             +----------------------------------+
    >18      ~             Reserved             ~
             ~                                  ~
             *----------------------------------*
        (>24 - maximum size)
```

19.2.12 Detach File (Subopcode >A1).

The format of the Detach File SVC call block is as follows:

```
                            Align on word boundary
 Hex Offset *----------------+-----------------*
    >00     |       00       | <Return Code>   |
            +----------------+-----------------+
    >02     |       A1       |    Reserved     |
            +----------------+-----------------+
    >04     ~              Reserved            ~
            +----------------------------------+
    >08     |            JSB Address           |
            +----------------------------------+
    >16     |         Pathname Pointer         |
            +----------------------------------+
    >18     ~              Reserved            ~
            ~                                  ~
            *----------------------------------*
       (>24 - maximum size)
```

If the JSB field is zero, the file is detached from the issuer's job. In order to specify another job, the issuing task must be software privileged, hardware privileged or a system task. The ROB associated with the JCB which corresponds to the given pathname is located. If more than one attach has been issued, the attach count is decremented and control is returned. If the attach count is zero, the ROB is deleted and the FCB luno count is decremented. When the luno count is zero the memory structures are released, and if the file was a temporary file, it is deleted.

19.2.13 Detach File by Number (Subopcode >A3).

The Detach File by Number SVC call block has the following format:

```
                              Align on word boundary
 Hex Offset  *----------------+-----------------*
    >00      |       00       |  <Return Code>  |
             +----------------+-----------------+
    >02      |       A3       | Detach Number   |
             +----------------+-----------------+
    >04      |             Reserved             |
             +----------------------------------+
    >06      |             Reserved             |
             +----------------------------------+
    >08      |            JSB Pointer           |
             +----------------------------------+
    >0A      ~             Reserved             ~
             ~                                  ~
             *----------------------------------*
         (>24 - maximum size)
```

The Detach File by Number SVC is issued by the Job Manager for each ROB that still exists for a job at its termination. Job Manager obtains the attach number from the ROB and places it into byte 3 of the Detach File by Number call block.

19.2.14 Modify FDR Bit (Subopcode >A4).

The Modify FDR Bit SVC is available to turn on or off a particular bit in an FDR. One of the flags in the call block indicates which bit to change and another flag indicates how to change that bit.

The format of the SVC call block is as follows:

```
                                     Align on word boundary
Hex Offset *-----------------+-----------------*
   >00     |       00        |  <Return Code>  |
           +-----------------+-----------------+
   >02     |       A4        |   Reserved      |
           +-----------------+-----------------+
   >04     | <System Flags>  |  User Flags     |
           +-----------------------------------+
   >06     ~             Reserved              ~
           ~                                   ~
           +-----------------------------------+
   >16     |         Pathname Pointer          |
           +-----------------------------------+
   >18     ~             Reserved              ~
           ~                                   ~
           *-----------------------------------*
      (>24 - maximum size)
```

```
    Byte 5 - user flags
         Bit    0    1=Set, 0=Clear specified bit
         Bit    1    1=Use temporary file bit
         Bits 2-7    Reserved, must be zero
```

19.2.15 Release LUNO in Another Job (Subopcode >A5).

The Release LUNO in Another Job SVC is used by the Job Manager and by PMTERM to clean up job-local and task-local LUNOs when a job or task terminates.

The format of the SVC call block is as follows:

```
                              Align on word boundary
 Hex Offset  *----------------+-----------------*
    >00      |       00       | <Return Code>   |
             +----------------+-----------------+
    >02      |       A5       | LUNO to Release |
             +----------------+-----------------+
    >04      | <System Flags> |    Reserved     |
             +----------------------------------+
    >06      |             Reserved             |
             +----------------------------------+
    >08      |  JSB of job from which to release|
             +----------------------------------+
    >0A      |  TSB of job from which to release|
             +----------------------------------+
    >0C      ~             Reserved             ~
             ~                                  ~
             +----------------------------------+
    >10      |          Utility Flags           |
             +----------------------------------+
    >12      ~             RESERVED             ~
             ~                                  ~
             *----------------------------------*
        (>24 - maximum size)
```

19.2.16 Assign System LUNO FF (Subopcode >A6).

The Assign System LUNO FF is used to create a logical device table (LDT) for a given program file, using its file control anchor (FCA). The format of the block is as follows:

```
                             Align on word boundary
Hex Offset *----------------+-----------------*
  >00      |       00       | <Return Code>   |
           +----------------+-----------------+
  >02      |       A6       |    Reserved     |
           +----------------+-----------------+
  >04      ~             Reserved             ~
           ~                                  ~
           +----------------------------------+
  >08      |           FMT Address            |
           +----------------------------------+
  >0a      |           FCB Address            |
           *----------------------------------*
      (>0C - maximum size)
```

19.2.17 Release File Structures (Subopcode >A7).

The Release File Structures operation is used by the I/O subsystem to remove file structures during certain abort conditions. It releases file control block (FCB), and the file directory block (FDB). The format of the block is as follows:

```
                                    Align on word boundary
   Hex Offset *-----------------+------------------*
     >00      |        00       |    Reserved      |
              +-----------------+------------------+
     >02      |        A7       |    Reserved      |
              +-----------------+------------------+
     >04      ~              Reserved              ~
              ~                                    ~
              +------------------------------------+
     >08      |            FMT Address             |
              +------------------------------------+
     >0A      |            FDB Address             |
              *------------------------------------*
         (>0C - maximum size)
```

19.2.18 DIOU Operations (Subopcodes >C2, >C3, >C6, >C7).

The DIOU operations use a call block described in a template named DCB. It is designed to simplify any buffering and unbuffering that has to be performed by placing the string field at the same location as the string field of the IRB (pathname field). The subopcodes that use the DIOU call block are:

>C2 - Get selected device parameters
>C3 - Set selected device parameters
>C6 - Get CDE From CDT
>C7 - Process device task bid

The DIOU Call Block (DCB) has the following format:

```
                         Align on Word Boundary
    Hex Offset  *--------------+--------------*
       >00      !      00      !<Return Code>!
                +--------------+--------------+
       >02      !  DCBOC       !  DCBLUN      !
                +--------------+--------------+
       >04      !  DCBSFL      !  DCBCDE      !
                +--------------+--------------+
       >06      !           Reserved          !
                +--------------+--------------+
       >08      !  DCBNAM      !              !
                +--------------+--------------+
                /              /              /
                /              /              /
                +--------------+--------------+
       >10      !            Reserved         !
                +--------------+--------------+
       >12      !           DCBNUM            !
                +--------------+--------------+
       >14      !           DCBUFL            !
                +--------------+--------------+
       >16      !           DCBBUF            !
                +--------------+--------------+

                ( >18 - maximum size)
```

```
 FLAGS FOR FIELD: DCBSFL     #04 - *SYSTEM FLAGS

   DCFBSY = (X...............) - BUSY
   DCFERR = (.X..............) - ERROR
```

```
FLAGS FOR FIELD: DCBUFL     #14 - *REQUESTOR FLAGS

  DCFCON = (X...............) - CONDITIONAL SET
  DCFNAM = (.X..............) - NAME SPECIFIED
  DCFRES = (..X.............) - RESERVED FLAG
  DCFWCH = (...XX...........) - WHICH RELATIVE DEVICE
  DCFREP = (.....X..........) - REPLACE
  DCFVOL = (......X.........) - VOLUME NAME PROVIDED
  DCFSDK = (.......X........) - USE SYSTEM DISK
```

EQUATES:

LABEL	EQUATE TO	VALUE	DESCRIPTION
DCBCHR	DCBCDE	>05	*BID CHARACTER

DCBOC

The operation code of the desired SVC is placed in this byte by the requester. The value returns unaltered.

DCBSFL

DCFBSY, when set to 1, indicates the SVC is being worked on. DCFERR, when set to 1, indicates an error was returned in DCBEC.

DCBCDE

All operations that require a CDE number (position within a CDT) get the number from this field. If the first CDE is desired this field is 0, second CDE is 1, and so on up to >F.

DCBCHR

DCBCHR is an equate for the DCBCDE field. The bid character on a bid task SVC is placed in this field.

DCBNAM

Up to eight character, left-adjusted, blank filled name of the device. This is provided by the requestor when DCFNAM is set to 1 for the SVCs that require a device name or number. It is filled in by DIOU when a device name is requested.

DCBNUM

The DIOU assigned number for the device. It is returned by DIOU when a device number is requested. It is provided by the user when DCFNAM is set to 0 for the SVCs that require a device name or number.

DCBUFL

DCFCON, when set to 1, indicates that the operation being performed is a Conditional Set Parameters. DCFNAM, when set to 1, indicates the name of the device is specified in the call block. When set to 0 the device number is specified. DCFWCH is a two bit flag that is used by the Get Device Parameters operation to indicate the parameters of the specified device are desired (00), or the parameters of the device that is lexically less than (01) or greater than (10) the specified device are desired. DCFREP is used when modifying a CDT. If DCFREP is set to 1, the specified CDE will replace the current CDE of that number. If DCFRGP is set to 0, the specified CDE will be added to the CDT only if none currently exists with the same CDE number. DCFVOL is set to 1 if a volume name instead of a disk name is provided in the call block. DCFSDK is set to 1 if the operation is applied to the system disk. The order in which DCFSDK, DSFVOL, and DCFNAM are checked is DCFSDK, DSFVOL, and then DCFNAM. In other words, DCFSDK overrides DCFVOL and DCFNAM, and DSFVOL overrides DCFNAM.

DCBBUF

The address of the device parameters or the CDE, depending on the subopcode. The address may be odd. When a buffer is used, the first byte contains the length of the buffer excluding the length byte.

19.2.18.1 >C2 - Get Selected Device Parameters.

The Get Selected Device Parameters operation return the values of the parameters identified in the buffer pointed to by the DCBBUF field. The name and number of the device are always returned in the call block when this operation completes successfully.

DCB fields used include: DCBOC (>C2), DCBNUM, DCBNAM, DCBUFL, and DCBBUF.

DCBNUM

If the DCFNAM flag (see DCBFLG) is 0, the parameters requested are those of the device specified by the device number in this field. If it is 1 the device number will be returned in this field even if it is not one of the parameters requested.

DCBNAM

If the DCFNAM flag is 1, the parameters requested are those of the device specified by the device name in this field. If it is 0, the device name will be returned in this field even if it is not one of the parameters requested.

DCBUFL

If the DCFNAM flag is 0, the number in the device number field identifies which device's parameters are requested; otherwise, the name in the device name field does. The DCFWCH flag field indicates which device's parameters relative to the specified one are to be returned. If the field is 00 use the specified device, 01 use the device that is lexically less than the specified device, or 10 use the device that is lexically greater than the specified device.

DCBBUF

The buffer pointed to by this field is set up by the user in the following manner.

```
  length of the buffer
  parameter number of a parameter
  parameter number of a parameter
                .
                .
                .
  parameter number of a parameter
                0
```

All of the values are bytes. The length of the buffer is the total number of bytes available for DIOU to place the specified parameters back into the buffer. If the name or number parameter is in the list of parameters the value will not be returned in the buffer but instead it will be placed in the name or number field of the call block. The buffer will be returned using the following format.

```
length of the buffer
parameter number of a parameter ----+
length of the parameter             |
parameter                           |- repeated
             -----------------------+
    .
    .
    .
    0
```

The length of the buffer will be the number of bytes taken up by the parameter overhead (one word per parameter) and the parameters. It will include a byte for the 0 terminator. If a parameter is not defined, the length byte for that parameter will be zero.

19.2.18.2 >C3 - Set Selected Device Parameters.

The Set Selected Device Parameters operation adds or changes the values of the parameters identified in the buffer pointed to by the DCBBUF field.

DCB fields used include: DCBOC (>C3), DCBNUM, DCBNAM, DCBUFL, and DCBBUF.

DCBNUM
: If the DCFNAM flag (see DCBFLG) is 0, the parameters to be set are those of the device specified by the device number in this field.

DCBNAM
: If the DCFNAM flag is 1, the parameters to be set are those of the device specified by the device name in this field.

DCBUFL
: If the DCFNAM flag is 0, the number in the device number field identifies which device's parameters are to be set, otherwise; the name in the device name field does. If the DCFCON flag is 1, the parameters in the list will be set if the verification value provided with the new value is the present value of the parameter.

DCBBUF
: If DCFCON is 0, the buffer pointed to by this field will be

set up by the user in the following form.

```
length of the buffer
parameter number of a parameter ----+
length of the parameter             |
parameter                           |- repeated
            ------------------------+
    .
    .
    .
    0
```

DIOU processes the buffer until it encounters a 0 parameter number or the end of the buffer as defined by the first byte of the buffer. The length byte does not include itself.

If DCFCON is 1, the buffer pointed to by this field is set up by the user in the following form.

```
length of the buffer
parameter number of a parameter ----+
length of the parameter             |
verification value                  |- repeated
parameter                           |
            ------------------------+
    .
    .
    0
```

The verification value must be the same length as the parameter. DIOU processes the buffer until it encounters a 0 parameter number or the end of the buffer as defined by the first byte of the buffer. The length byte does not include itself. Note that only those parameters not declared as READ ONLY may be altered.

19.2.18.3 >C6 - Get CDE From CDT.

The Get CDE From CDT operation returns the requested command definition entry from the specified command definition table.

DCB fields used include: DCBOC (>C6), DCBCDE, DCBNUM, and DCBBUF.

DCBCDE
This field contains a value between 0 and >F that identifies the CDE to be retrieved.

DCBNUM
The CDT number from which the CDE is to be retrieved is placed in this field. The first CDT is number 0.

DCBBUF

The CDE is returned in the buffer pointed to by this field. The first byte will be the length of the CDE.

19.2.18.4 >C7 - Process Device Task Bid.

The Process Device Task Bid operation processes the CDE corresponding to the character passed in the call block if the character is in the specified terminal's CDEs.

DCB fields used include: DCBOC (>C7), DCBCDE, DCBNUM, DCBNAM, and DCBUFL.

DCBCDE

The character of a CDE.

DCBNUM

If the DCFNAM flag (see DCBFLG) is 0, the task bid is from the device specified by the device number in this field.

DCBNAM

If the DCFNAM flag is 1, the task bid is from the device specified by the device name in this field.

DCBUFL

If the DCFNAM flag is 0, the number in the device number field identifies which device the task is bid from; otherwise, the name in the device name field identifies the device.

19.3 SPECIAL FEATURE OF EXECUTE TASK SVC

The Execute Task SVC (>2B) uses a bit in the flag byte of the call block for implementing the RBID function. When the bit is set, the SVC processor bids the task and unconditionally suspends the caller. The caller is placed in state 6, as opposed to state 17. This allows the task that is bid and the task that bid it to alternate execution. When the task that was originally bid terminates, the caller is reactivated.

The sole function of the RBID bit is to support the SCI RBID capability. The RBID process is described in detail in the DNOS SCI and Utilities Design Document.

19.4 SEGMENT MANAGEMENT

The Reset Exclusive Use Across Job Boundaries suboperation (>13) of the Segment Management SVC (>40) is used by the task termination processor (PMTERM) to clean up any segments still owned at task termination. PMTERM will continue to issue this SVC as long as there are owned segment entries (OSEs) linked to the TSB. This operation uses the same SVC processor as the Reset Exclusive Use suboperation but uses a different entry point. If the task in which exclusive use is being reset is not in state 04 (task termination), the SVC will fail. Otherwise, the segment owner block (SOB) is delinked from the SSB and its memory released. The OSE is removed from the list of owned segments linked to the TSB and its memory released. If the segment is not in use or reserved, it is cached or deleted.

19.5 NAME MANAGEMENT

The Name Management SVC (SVC >43) supports 15 subopcodes, most of which are not useful to user programs. Four of the operations are documented in the DNOS SVC Reference Manual. These are:

* Return the pathname and parameters for a logical name

* Create a logical name (Setting a name's value)

* Delete a logical name

* Restore name segment

The following operations are documented only in this section, since they are useful only to operating system and DNOS utility tasks:

* Return an additional pathname of a set of pathnames for a logical name

* Add a pathname to the set of pathnames for a logical name

* Return the logical name that immediately precedes or follows in alphabetical order the specified logical name in the current stage

* Delete a defined subset of the logical names in the current stage

* Create a new stage

- * Return to the previous stage
- * Return error information
- * Return segment size
- * Copy logical names to new segment
- * Create empty name segment
- * Save name segment

A task that requires a new logical name performs a Set Name's Value (subopcode >02) operation, supplying the logical name, the value (pathname), and the parameters, if any. When the value of a logical name is a set of pathnames, the task performs an Append Pathname to Name (subopcode >03) operation, supplying the logical name and a pathname. The append operation is performed for each additional pathname.

When a task needs the pathname or parameters of a logical name, it performs a Determine Name's Value (subopcode >00) operation. When the value of the logical name is a set of pathnames, the task performs a Determine Next Pathname (subopcode >01) operation for each additional pathname. The task supplies the logical name for either of these two operations. Within DNOS, most Assign LUNO operations go through the Name Manager for resolution of names, therefore a task rarely needs to use this operation.

The Delete Name (subopcode >04) operation deletes a specified logical name. The Return Next Name (subopcode >05) operation returns a logical name that is adjacent to the supplied name in the current name list. A flag specifies the preceding name or the following name.

The Purge Names (subopcode >06) operation supplies a logical name and its length. The name is compared to each logical name in the current name list. If an equal name is found, that name is deleted. When all but the rightmost character of the names are equal, and the rightmost character of the name in the table is greater than the corresponding character of the specified name, the operation deletes the name.

The Enter New Stage (subopcode >07) operation creates a new stage with the calling task as the only task in the stage. The Return to Previous Stage (subopcode >08) operation may only be executed by the task that created the current stage. It returns that task to the previous stage of the task. It deletes the stage if it has no daughter stage and if the calling task is the only task in the stage.

The Return Next Error Entry (subopcode >09) returns the values of error synonyms stored in the first entry in the list of error entries.

Several operations have to do with the segments that contain logical names and synonyms. The Determine Segment Size (subopcode >0A) operation returns the size required for a segment that would hold the names available to the calling task. The Copy Names to New Segment (subopcode >0B) operation copies all names available to the calling task to the specified segment. The Create Empty Name Segment (subopcode >0D) creates a new segment for names and Save Name Segment (subopcode >0E) copies the name segment to disk. The Restore Name Segment (subopcode >0C) retrieves a disk copy of a name segment into memory.

The supervisor call block is of the following form:

```
                                   Align on word boundary
Hex Offset *--------------------+--------------------*
    >00    |        >43         |   <Return Code>    |
           +--------------------+--------------------+
    >02    |     Subopcode      |       Flags        |
           +--------------------+--------------------+
    >04    |             Address of Name             |
           +-----------------------------------------+
    >06    |             Address of Value            |
           +-----------------------------------------+
    >08    |        Address of Parameter List        |
           +-----------------------------------------+
    >0A    |              Miscellaneous              |
           +-----------------------------------------+
    >0C    |                Reserved                 |
           *-----------------------------------------*
```

The call block contains the following:

Byte	Contents
0	Opcode, >43.
1	Return code. DNOS returns zero when the operation completes satisfactorily. When the operation completes in error, DNOS returns an error code.

2 Subopcode, as follows:

>00 - Determine Name's Value
>01 - Determine Next Pathname
>02 - Set Name's Value
>03 - Append Pathname to Name
>04 - Delete Name
>05 - Return Next Name
>06 - Purge Names
>07 - Enter New Stage
>08 - Return to Previous Stage
>09 - Return Next Error Entry
>0A - Determine Segment Size
>0B - Copy Names to New Segment
>0C - Reserved
>0D - Create Empty Name Segment
>0E - Save Name Segment
>0F - Restore Name Segment

3 Flags:

Bit 0 - Name type. Set as follows:
    1 - Logical Name
    0 - Synonym
Bits 1-7 - Reserved

4-5 Address of name. Address of a buffer that contains the name (or name list for the return to previous stage operation). The first byte of the buffer contains the length of the name (or name list).

6-7 Address of value. Address of a buffer that contains a pathname or other value for a logical name. The first byte of the buffer contains the length of the pathname.

8-9 Address of parameter list. Address of a buffer that contains a parameter list for a logical name. The first byte of the buffer contains the length of the list.

10-11 Miscellaneous. Used as a flag by the determine name's value and the return next name operations. Used for the parameter number by the determine next pathname operation. Used for the segment size by the determine segment size operation. Used for segment ID by the copy names to new segment operation.

12-13 Reserved.

The task that requests a Set Name's Value operation for a logical name with parameters must supply the parameters in a list. The Determine Name's Value operation returns a parameter list also when the name is a logical name having a parameter list. The format of the parameter list is as follows:

```
Hex Offset *-------------------+--------------------* --------
    00     |        Length     |          0         |
           +-------------------+--------------------+
    02     | Type for Sublist  | Length of Sublist  |
           +-------------------+--------------------+ Required
    04     |                                        |
           ~               Parameter                ~
           ~             Entry Blocks               ~
    2+n    |                                        |
           +-------------------+--------------------+ --------
    4+n    | Type for Sublist  | Length of Sublist  |
           +-------------------+--------------------+
    6+n    |                                        | Optional
           ~               Parameter                ~
           ~             Entry Blocks               ~
    4+n+m  |                                        |
           *----------------------------------------* --------
```

The parameter list contains the following:

Byte Contents

0 Length. Length of entire structure; the sum of 1 plus twice the number of sublists.

1 Zero.

One or more of the following sublists:

2 Type for sublist. The type of the parameters in the sublist. Types of parameters are:

0 - System parameters
1 - Spooler parameters
2->7F - Reserved
>80->FF - User IPC parameters

3 Length of sublist. The sum of the lengths of all parameter entry blocks in the sublist, referred to as n.

4-3+n Parameter entry blocks, one for each parameter. Formats of parameter entry blocks are described in subsequent paragraphs.

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

```
*----------------+-+-*
| Parameter No.  |1|V|
*----------------+-+-*
```

The parameter entry block contains the following:

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

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

```
*----------------+-+-*
| Parameter No.  |0|0|
+----------------+-+-+
|       Value        |
*--------------------*
```

The parameter entry block contains the following:

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

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

```
*----------------+-+-*
| Parameter No.  |0|1|
+----------------+-+-+
|  Parameter Length  |
+--------------------+
|                    |
~      Parameter     ~
~        Value       ~
|                    |
*--------------------*
```

The parameter entry block contains the following:

Byte	Contents
0	Parameter number byte: Bits 0-5 - Parameter number, 0 through 63. Parameter numbers need not be assigned or ordered in sequence, but must be unique within the sublist. Bit 6 - 0 Bit 7 - 1
1	Parameter length. The number of bytes required for the parameter value.
2-nn	Parameter value. The numbers or characters of the parameter.

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

19.5.1 Determine Next Pathname (Subopcode >01).

When the Determine Name's Value operation returns 1 in the miscellaneous field, the task executes one or more Determine Next Pathname operations (subopcode >01). Each operation returns a pathname and indicates whether or not there is another pathname.

The following fields of the supervisor call block apply:

- Opcode - >43
- Return code
- Subopcode - >01
- Flags
- Address of name
- Address of value
- Miscellaneous

The operation is defined only for logical names. The flag for name type must be set to 1.

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

The address of value field must contain the address of a buffer large enough to contain the pathname expected. The first byte in the buffer contains the length of the buffer. The operation replaces that value with the length of the pathname, and places the characters of the pathname in succeeding bytes.

The miscellaneous field contains the number of the next pathname (the pathname to be returned). Pathname 0 is the first pathname, pathname 1 is the second, etc. The operation increments the number, or sets the field to zero when the last pathname is returned.

19.5.2 Append Pathname to Name (Subopcode >03).

When a logical name represents a set of pathnames, the Append Pathname to Name operation (subopcode >03) adds the pathnames, one at a time. The logical name must have been created, and must have a pathname and parameters (if required) prior to performing this operation.

The following fields of the supervisor call block apply:

- Opcode - >43
- Return code
- Subopcode - >03
- Flags

* Address of name

* Address of value

The operation is defined only for logical names. The flag for name type must be set to 1.

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

The address of value field must contain the address of a buffer that contains an additional pathname. The first byte in the buffer contains the length of the pathname. Successive bytes contain the characters of the pathname. Pathnames for files being concatenated must be unique.

19.5.3 Return Next Name (Subopcode >05).

The Return Next Name operation (subopcode 05) returns the next logical name and, optionally, the pathname of the logical name. The next logical name is either the preceding or following name in alphabetic order, selected by a flag supplied in the call block.

The following fields of the supervisor call block apply:

* Opcode - >43

* Return code

* Subopcode - >05

* Flags

* Address of name

* Address of value

* Miscellaneous

When the name type flag in the flags byte is set to zero, the operation expects the name to be a synonym, and returns a synonym.

The address of name field must contain the address of a buffer that contains the length of a name in the first byte, and the characters of that logical name in succeeding bytes. The buffer may contain zero in the first byte when no logical name is supplied. The operation returns the first name in alphabetical order when no logical name is supplied.

The operation returns the next logical name in the buffer at the address in the address of name field. When the next logical name does not exist, the operation returns a zero in the first byte of the buffer. This buffer must be large enough to contain an eight-character logical name.

The address of value field must contain the address of a buffer in which the operation returns the pathname. When the calling task places zero in the first byte of this buffer, the operation does not return a pathname. The buffer must be large enough to contain the longest pathname that could be returned.

The contents of the miscellaneous field determines which name is the next name. When the field contains 1, the operation returns the preceding name in alphabetical order. When the field contains 0, the operation returns the succeeding name.

19.5.4 Purge Names (Subopcode >06).

The Purge Names operation (subopcode >06) deletes related logical names from the set of logical names available to the task. The calling task supplies a name containing N characters. A name is deleted if any of the following statements is true of that name:

* The name contains N characters, and these characters are equal to those of the supplied name.

* The name contains N characters, all but the last are equal to those of the supplied name, and the last character is greater than the last character of the supplied name.

* The name contains more than N characters and the first N characters are equal to those of the supplied name.

* The name contains more than N characters, the first N-1 characters are equal to the corresponding characters of the supplied name, and the Nth character is greater than the Nth character of the supplied name.

No name that is shorter than the supplied name is deleted. Otherwise, the length of the name does not determine whether or not it is deleted. The pathname of the name is ignored in selecting a name for deletion.

The following fields of the supervisor call block apply:

* Opcode - >43

* Return code

* Subopcode - >06

- * Flags
- * Address of name

When the name type flag in the flags byte is set to zero, the operation expects the name to be a synonym, and purges the synonyms available to the calling task.

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

As an example, the following logical names are defined for a task:

INFILE	OUTFILE	INPUT
ABCD	ABCDE	ABCDEF

If a purge names operation were performed supplying name IN, names INFILE and INPUT would be deleted. If the name supplied were INF, the same two names would be deleted. If ABCDE were supplied as a name, names ABCDE and ABCDEF would be deleted.

19.5.5 Enter New Stage (Subopcode >07).

When the Enter New Stage operation (subopcode 07) is issued, the calling task becomes the first and only task in a new stage. Tasks bid by the calling task also execute in the new stage. Logical names and synonyms available to the task prior to entering the new stage become the logical names and synonyms for the new stage. Additions and changes made by tasks in the stage apply only to the stage.

The following fields of the supervisor call block apply:

- * Opcode - >43
- * Return code
- * Subopcode - >07

19.5.6 Return to Previous Stage (Subopcode >08).

Only a task that has created a new stage may return to the previous stage (the stage it belonged to before creating the new stage). The Return to Previous Stage operation (subopcode >08) returns the calling task to the previous stage, optionally taking a set of synonym names.

When other tasks remain in the current stage, or when descendant stages remain, the calling task returns to the previous stage, and the current stage remains. When the calling task is the only task in the current stage and no descendant stages exist, the calling task returns to the previous stage and the current stage is deleted.

The descendant error list is the mechanism for retaining the values of error synonyms $$CC, $$ES, $$MN, $$FN, and $$VT when a stage is deleted. When a descendant stage is deleted, and that stage has error synonym $$CC in the synonym table, a descendant error list entry is built. The entry becomes the first entry in the descendant error list for the stage, or an additional entry in an existing list.

The descendant error list consists of a byte that contains the length of the list, followed by the entries of the list. Each entry consists of the following:

* The value of error synonym $$CC, preceded by a byte that contains the length of the value.

* The value of error synonym $$ES, preceded by a byte that contains the length of the value, or zero if no value has been assigned.

* The value of error synonym $$MN, preceded by a byte that contains the length of the value, or zero if no value has been assigned.

* The value of error synonym $$FN, preceded by a byte that contains the length of the value, or zero if no value has been assigned.

* The value of error synonym $$VT, preceded by a byte that contains the length of the value, or zero if no value has been assigned.

When a stage is deleted, the following actions occur:

* Add the entries, if any, in the descendent error list of the terminating stage to the list of the previous stage.

* When error synonym $$CC has been assigned for the terminating stage, place a descendant error list entry in the list of the previous stage.

* Delete synonyms and logical names defined for the terminating stage.

The following fields of the supervisor call block apply:

* Opcode - >43

* Return code

* Subopcode - >08

* Address of name

The address of name field contains the address of a buffer, or zero. When the field contains zero, no synonym names are returned. When the field contains an address, the address is the address of a synonym buffer. The buffer contains a byte that contains the length of the synonym list, followed by the names of the synonyms in the list. Each synonym is preceded by a byte that contains the length of the synonym. The synonym buffer is returned to the previous stage, and the Name Manager adds the synonyms to those of the previous stage, obtaining the values from the synonym segment.

19.5.7 Return Next Error Entry (Subopcode >09).

The Return Next Error Entry operation (subopcode >09) obtains the first entry on the descendant error list previously described. The operation also deletes the entry. This operation is normally performed by system tasks that process errors.

The following fields of the supervisor call block apply:

* Opcode - >43

* Return code

* Subopcode - >09

* Address of value

The address of value field contains the address of a buffer in which the operation returns a descendant error list entry. The first byte of the buffer contains the length of the buffer. When there is no entry, the operation returns zero in the first byte of the buffer. Otherwise, the operation returns the length of the entry in the first byte of the buffer, followed by the characters of the entry.

19.5.8 Determine Segment Size (Subopcode >0A).

The Determine Segment Size operation (subopcode >0A) returns the size required for a segment large enough to contain all the logical names or synonyms available to the calling task.

The following fields of the supervisor call block apply:

* Opcode - >43

* Return code

* Subopcode - >0A

* Flags

* Miscellaneous

The name type flag in the flags field is set to 0 to obtain the size for a synonym segment, or to 1 to obtain the size for a logical name segment.

The size, in bytes, required for the specified name segment is returned in the miscellaneous field.

19.5.9 Copy Names to New Segment (Subopcode >0B).

The Copy Names to New Segment operation (subopcode >0B) copies the logical names or synonyms available to the calling task to a specified segment. The segment size should be obtained using the Determine Segment Size operation, and a segment of that size should be allocated. The segment should be a memory-based segment with the share-protect and reusable attributes.

The following fields of the supervisor call block apply:

* Opcode - >43

* Return code

* Subopcode - >0B

* Flags

* Miscellaneous

The name type flag in the flags field is set to 0 to copy a synonym segment, or to 1 to copy a logical name segment. The segment ID of the allocated segment is placed in the miscellaneous field.

19.5.10 Creating an Empty Name Segment (Subopcode >0D).

The Create Empty Name Segment operation (subopcode >0D) creates an empty logical name segment. The following fields of the supervisor call block apply:

* Opcode - >43

* Return Code

* Subopcode - >0D

- * Flags
- * Segment size
- * Segment ID

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

The address of name field, address of value field, and address of parameter must be zero.

The segment size is a value in bytes, specifying the initial size of the segment. The run ID of the segment is returned.

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

```
       EVEN                 CREATE EMPTY NAME SEGMENT
CEMNAM BYTE >43
CENS   BYTE 0
       BYTE >0D
       BYTE >00
       DATA 0
       DATA 0
       DATA 0
       DATA >400
CID    DATA 0
```

19.5.11 Saving a Name Segment (Subopcode >0E).

The Save Name Segment operation (subopcode >0E) saves a logical name segment to a disk file. The following fields of the supervisor call block apply:

- * Opcode - >43
- * Return Code
- * Subopcode - >0E
- * Flags
- * Address of name
- * Segment ID

The only flags examined are the global flag and the run ID flag. The global flag set to one indicates that the global names are to be saved. The run ID flag indicates that the segment to save is passed in the call block in the run ID field. If no flags are set the job local synonyms are saved.

The address of name field must contain the address of a buffer that contains the length of the name in the first byte and the characters of the logical name definition in succeeding bytes. Names can be saved on any disk on any system. The one restriction is job-local logical names may not be used to specify the pathname.

The address of value field, and address of parameter list field must be zero.

If the user flag for global operation is set to one, the global names will be written, otherwise the job local names will be written, unless the run ID flag is set to one, in which case the segment passed in the run ID field will be saved.

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

```
       EVEN                SAVE NAME SEGMENT
SAVNAM BYTE >43
SVNS   BYTE 0
       BYTE >0E
       BYTE >00
       DATA LNME
       DATA 0
       DATA 0
LNME   BYTE 11
       TEXT '.DISK.FILE2'
```

19.6 MODIFY BTA OR JCA SIZE

The Modify BTA or JCA Size SVC (>4A) is of the following format:

```
                          Align on word boundary
                          Software privileged
Hex Offset *---------------+-----------------*
    00     |      4A       | <Return Code>   |
           +---------------+-----------------+
    02     | Subopcode     |   Reserved      |
           +---------------+-----------------+
    04     |    Number of Beets Requested    |
           +---------------------------------+
    06     | Segment Run ID (subopcode 3)    |
           *---------------------------------*
```

Byte 2 - subopcodes:

0 - Allocate more static buffer
1 - Deallocate static buffer
2 - Find amount of static buffer currently in use
3 - Expand the segment ID specified

Subopcode 0 of this SVC is used to expand the BTA used by the I/O subsystem. Memory is taken from dynamic user memory to expand the BTA as one contiguous block at the end of system memory.

Subopcode 1 is used to release memory (number of beets specified) from the BTA and return it to dynamic user memory. Subopcode 2 retrieves the number of beets of BTA currently in use. Subopcode 3 is used to expand the size of a JCA segment that is currently in memory but is not a memory-resident segment. The segment is expanded by the amount specified in bytes 4 and 5 of the call block. The new size must be less than >800 beets and it must be less than the maximum JCA size.

19.7 HALT/RESUME TASK

The Halt/Resume Task SVC is of the following format:

```
                             Align on word boundary
                             Software privileged
  Hex Offset *----------------+-----------------*
      00     |       4B       |  <Return Code>  |
             +----------------+-----------------+
      02     | Task Run ID    | Task Station ID |
             +----------------+-----------------+
      04     | <Task State>   |  Sub-Opcode     |
             +----------------------------------+
      06     |             Reserved             |
             *----------------------------------*
```

Byte 5 - Subopcodes:

0 - Halt task
1 - Resume task

The Halt/Resume Task SVC is used by the SCI Debugger to halt a task before showing debugging information and to resume the task once that information has been displayed to the user.

The program management routine PMHALT executes this SVC at XOP level. If the station ID supplied is nonzero, the routine first checks to be sure a task with the supplied run-time ID is running at that station. If no such task is running, an error code of 1 is returned.

If a task is found, the current state of the task is returned in the byte reserved for the task state. If the subopcode requested is Halt, PMHALT sets the halt bit in the TSB flags TSBFL2. If the current task state is active, PMHALT calls the deactivate routine and sets the task state to 6 (unconditional wait).

If the subopcode requested is Resume, PMHALT clears all of the debug (breakpoint) bits and the halt bit in the TSB flags TSBFL2. If the task is in state 6, PMHALT calls NFPACT to place the task on the active queue. In other cases, it exits after setting the TSB flags.

19.8 EXPAND JCA

The job communication area (JCA), one of the data structures maintained for each job by the system, is expandable. The system uses the Expand JCA operation to provide more space for the JCA as required. This operation is not available to user tasks.

Subopcode >08 specifies the Expand JCA operation. Only the first 16 bytes of the supervisor call block apply. The specific fields are:

* Opcode

* Return code

* Subopcode

* Job run ID

The job run ID is supplied by the system task.

The following is an example of coding for a supervisor call block for an Expand JCA operation:

```
       EVEN               EXPAND THE JCA FOR JOB >4F.
EXPJCA BYTE >48
XJERR  BYTE 0
       BYTE >08
       BYTE 0
       DATA 0
       DATA >4F
       DATA 0,0,0,0
```

SECTION 20

LINKING INFORMATION FOR DNOS

20.1 OVERVIEW

Linking the portions of DNOS involves standard use of the Link Editor for assembly language and for Pascal modules. Link control files for the nonconfigurable portions of DNOS are found in the directories DSC.LINK.SYSTEM and DSC.LINK.UTILITY. Configurable portions of DNOS are determined during system generation and appropriate selections are included in the link control files DMLINK, IOULINK, and SYSLINK.

Conventions used in building the link control files are simple. Each file makes use of the NOPAGE and ERROR options of the Link Editor. Wherever possible LIBRARY and INCLUDE statements are used to keep the files short and easy to read.

Tasks written in assembly language generally include only their own object library. If they make use of S$ routines, they include the library VOLOBJ.SCI990.S$OBJECT. In addition to the S$ library, tasks written in Pascal have three run-time libraries available to them. These are located via the PASCAL synonym in the libraries PASCAL.MINOBJ, PASCAL.LUNOBJ, and PASCAL.OBJ. The MINOBJ library is a minimum size library for a non-debug environment. The LUNOBJ library is for a non-SCI environment and includes routines to support I/O by LUNO. The OBJ library is the run-time collection for either environment.

Pascal tasks which make use of standard initialization must use the statement INCLUDE (R$TASKDP) as the first item in the link control stream after the TASK or PHASE 0 statement. When linking a Pascal task which has a single procedure segment, the PROCEDURE declaration must be followed by INCLUDE (R$PROCDP). If a procedure is shared by more than one task, a SEARCH command must appear at the end of the include statements for the procedure, and an ALLOCATE command must immediately follow the R$TASKDP include statement. This is to ensure that all the shareable run-time routines are included in the procedure segment.

The directory DSC.LINK.SYSTEM includes link control files for each of the DNOS system tasks. The directory DSC.LINK.UTILITY includes link control files for each of the utility tasks. The directory DSC.LINK.DSR has link control files for DSRs, and the directory DSC.LINK.DNOSPROG has link control files for the programs used to build DNOS. Linkmaps for each of these are found in the directories VOLLST.LINKMAP.SYSTEM and VOLLST.LINKMAP.UTILITY, where VOLLST is the volume to which listings are written during a build of DNOS. The .SYSTEM directory includes linkmaps for each of the DNOS system tasks; the .UTILITY subdirectory for each of the utility tasks which supports SCI commands. To find the appropriate linkmap for a given SCI command, check the task name being bid by the command procedure and search for that name as a linkmap file name in the DSC.LINKMAP.UTILITY directory.

The paragraphs below describe the linking information from link control files for a system task written in assembly language, a system task written in Pascal, a DSR, and the system nucleus (seed). Examples of link control files generated during system generation are also presented.

20.2 LINKING A SYSTEM TASK

The link control stream for a system task must provide for the task to be loaded at address >C000. This requires use of the PHASE command with the base option set to >C000. The starting address of >C000 allows the task address space to include the system root and a JCA prior to the task code. All system tasks are linked with the procedure DUMROOT, which enables access to the system root. DUMROOT is built from the interrupt tables, the system seed, and the common blocks initialized by sysgen.

Appropriate libraries must be specified if subroutine support modules are included. The link control stream for the IPL task is shown below. It includes support routines from the DISKMGR and UTCOMN directories as well as the set of IPL object modules. The synonym VOLOBJ is used throughout this section to represent the volume on which the operating system object directories reside.

```
NOPAGE
ERROR
FORMAT IMAGE,REPLACE
LIBRARY    VOLOBJ.DISKMGR.OBJECT
LIBRARY    VOLOBJ.UTCOMN.OBJECT
LIBRARY    VOLOBJ.NAMMGR.OBJECT
PROCEDURE DUMROOT
DUMMY
INCLUDE     VOLOBJ.S$SGU$.DUMROOT
PHASE 0,IPL,PROG >C000
INCLUDE     VOLOBJ.LOADERS.OBJECT.SLIPL
INCLUDE     VOLOBJ.LOADERS.OBJECT.SLCRSH
INCLUDE     VOLOBJ.LOADERS.OBJECT.SLDINT
INCLUDE     VOLOBJ.LOADERS.OBJECT.SLDIO
INCLUDE     VOLOBJ.LOADERS.OBJECT.SLDISP
INCLUDE      VOLOBJ.LOADERS.OBJECT.SLDSR
INCLUDE      VOLOBJ.PERFORM.OBJECT.PFWCSO
INCLUDE      VOLOBJ.PERFORM.MOBJECT.PFDWCS
INCLUDE      VOLOBJ.LOADERS.OBJECT.SLFDB
INCLUDE      VOLOBJ.LOADERS.OBJECT.SLINIT
INCLUDE      VOLOBJ.LOADERS.OBJECT.SLIV
INCLUDE      VOLOBJ.LOADERS.OBJECT.SLJCA
INCLUDE      VOLOBJ.LOADERS.OBJECT.SLLMOD
INCLUDE      VOLOBJ.LOADERS.OBJECT.SLOPEN
INCLUDE      VOLOBJ.LOADERS.OBJECT.SLPFIO
INCLUDE      VOLOBJ.LOADERS.OBJECT.SLTABL
INCLUDE      VOLOBJ.LOADERS.OBJECT.SLTASK
INCLUDE      VOLOBJ.LOADERS.OBJECT.SLVRFY
INCLUDE      VOLOBJ.LOADERS.OBJECT.SLWCS
INCLUDE     (UTPTCH)
INCLUDE     (UTVERS)
INCLUDE      VOLOBJ.LOADERS.OBJECT.SLEND
END
```

The following link control stream links the log formatter task, LGFORM. It is written in Pascal and requires run time support, S$ routine support, interface with assembly language routines using the PASASM directory, and routines from the UTCOMN directory. The file specifies the Pascal base routine R$TASKDP as the first portion of the LGFORM task and includes required modules from the various directories as needed.

```
NOPAGE
NOSYMT
LIBRARY     VOLOBJ.LOG.OBJECT
LIBRARY     PASCAL.MINOBJ,PASCAL.LUNOBJ,PASCAL.OBJ
LIBRARY     VOLOBJ.UTCOMN.OBJECT
LIBRARY     VOLOBJ.PASASM.OBJECT
PROCEDURE DUMROOT
DUMMY
INCLUDE     VOLOBJ.S$SGU$.DUMROOT
PHASE 0,LGFORM,PROG >C000      ; SYSTEM LOG FORMATTER TASK
INCLUDE (R$TASKDP)
INCLUDE (LGFORM)
INCLUDE (LGDATA)
INCLUDE (UTR$ST)
INCLUDE (UTPTCH)
END
```

20.3 LINKING A DSR

A DSR must be linked as a system task, including the relevant source code modules written by the user as well as the appropriate modules from VOLOBJ.IOMGR.OBJECT to access support subroutines. The section on writing DSRs lists the location and function of each of the support subroutines. The example DSR link given below is for the 911 VDT and includes the modules which provide keyboard support (IOKB) and end of record processing (IONRCD).

```
NOPAGE
ERROR
PROCEDURE DUMROOT
DUMMY
INCLUDE      VOLOBJ.S$SGU$.DUMROOT
PHASE 0,DSR911,PROG >C000
INCLUDE      VOLOBJ.DEVDSR.OBJECT.DSR911
INCLUDE      VOLOBJ.IOMGR.OBJECT.IONRCD
INCLUDE      VOLOBJ.IOMGR.OBJECT.IOKB
INCLUDE      VOLOBJ.UTCOMN.OBJECT.UTPTCH
INCLUDE      VOLOBJ.DEVDSR.OBJECT.PWRFY
END
```

20.4 LINKING THE DNOS SEED

The common nucleus functions are linked into a task named SEED using the link control file in DSC.LINK.SYSTEM.SEED. Modules are included if they must be used by operating system code in any of a number of mapping configurations. The SEED is included as part of the system root when the system is generated. The following is an example of the link control file for the SEED.

```
ERROR
PARTIAL
TASK   SEED
INCLUDE      VOLOBJ.NUCLEUS.OBJECT.NFMAT
INCLUDE      VOLOBJ.IOMGR.OBJECT.IOBM
INCLUDE      VOLOBJ.IOMGR.OBJECT.IODBGN
INCLUDE      VOLOBJ.IPC.OBJECT.IPCPR
INCLUDE      VOLOBJ.IPC.OBJECT.IPCXFR
INCLUDE      VOLOBJ.LOG.OBJECT.LGQACC
INCLUDE      VOLOBJ.LOG.OBJECT.LGQBLK
INCLUDE      VOLOBJ.NUCLEUS.OBJECT.NFACTL
INCLUDE      VOLOBJ.NUCLEUS.OBJECT.NFACTQ
INCLUDE      VOLOBJ.NUCLEUS.OBJECT.NFATOL
INCLUDE      VOLOBJ.NUCLEUS.OBJECT.NFCLOK
INCLUDE      VOLOBJ.NUCLEUS.OBJECT.NFCOPY
INCLUDE      VOLOBJ.NUCLEUS.OBJECT.NFCRSH
INCLUDE      VOLOBJ.NUCLEUS.OBJECT.NFDACT
INCLUDE      VOLOBJ.NUCLEUS.OBJECT.NFCEOR
INCLUDE      VOLOBJ.NUCLEUS.OBJECT.NFDEF
INCLUDE      VOLOBJ.NUCLEUS.OBJECT.NFDLNK
INCLUDE      VOLOBJ.NUCLEUS.OBJECT.NFDLOV
INCLUDE      VOLOBJ.NUCLEUS.OBJECT.NFDOOR
INCLUDE       VOLOBJ.NUCLEUS.OBJECT.NFDOVB
INCLUDE       VOLOBJ.NUCLEUS.OBJECT.NFDQ1
INCLUDE       VOLOBJ.NUCLEUS.OBJECT.NFDQH
INCLUDE       VOLOBJ.NUCLEUS.OBJECT.NFDTOL
INCLUDE       VOLOBJ.NUCLEUS.OBJECT.NFDTSK
INCLUDE       VOLOBJ.NUCLEUS.OBJECT.NFDWOM
INCLUDE       VOLOBJ.NUCLEUS.OBJECT.NFDWOT
INCLUDE       VOLOBJ.NUCLEUS.OBJECT.NFENAB
INCLUDE       VOLOBJ.NUCLEUS.OBJECT.NFEOR
INCLUDE       VOLOBJ.NUCLEUS.OBJECT.NFINT2
INCLUDE       VOLOBJ.NUCLEUS.OBJECT.NFLOVB
INCLUDE       VOLOBJ.NUCLEUS.OBJECT.NFLWCS
INCLUDE       VOLOBJ.NUCLEUS.OBJECT.NFMAPO
INCLUDE       VOLOBJ.NUCLEUS.OBJECT.NFPACT
INCLUDE       VOLOBJ.NUCLEUS.OBJECT.NFPOP
INCLUDE       VOLOBJ.NUCLEUS.OBJECT.NFPSH
INCLUDE       VOLOBJ.NUCLEUS.OBJECT.NFPWOT
INCLUDE       VOLOBJ.NUCLEUS.OBJECT.NFPWUP
INCLUDE       VOLOBJ.NUCLEUS.OBJECT.NFQERR
INCLUDE       VOLOBJ.NUCLEUS.OBJECT.NFQOVB
INCLUDE       VOLOBJ.NUCLEUS.OBJECT.NFQUE1
INCLUDE       VOLOBJ.NUCLEUS.OBJECT.NFQUEH
INCLUDE       VOLOBJ.NUCLEUS.OBJECT.NFSRTN
INCLUDE       VOLOBJ.NUCLEUS.OBJECT.NFTBDO
INCLUDE       VOLOBJ.NUCLEUS.OBJECT.NFTERM
INCLUDE       VOLOBJ.NUCLEUS.OBJECT.NFTMGR
INCLUDE       VOLOBJ.NUCLEUS.OBJECT.NFTRTN
INCLUDE       VOLOBJ.NUCLEUS.OBJECT.NFWAKE
INCLUDE       VOLOBJ.NUCLEUS.OBJECT.NFWOMJ
INCLUDE       VOLOBJ.NUCLEUS.OBJECT.NFWOML
INCLUDE       VOLOBJ.NUCLEUS.OBJECT.NFWOTL
INCLUDE       VOLOBJ.NUCLEUS.OBJECT.NFXOPS
INCLUDE       VOLOBJ.NAMMGR.OBJECT.NMTRAN
INCLUDE       VOLOBJ.PROGMGR.OBJECT.PMMPRI
INCLUDE       VOLOBJ.PROGMGR.OBJECT.PMTSCH
INCLUDE       VOLOBJ.REQPROC.OBJECT.RPDQ1
INCLUDE       VOLOBJ.REQPROC.OBJECT.RPMAP2
INCLUDE       VOLOBJ.REQPROC.OBJECT.RPPRCK
INCLUDE       VOLOBJ.REQPROC.OBJECT.RPSGCK
INCLUDE       VOLOBJ.SEGMGR.OBJECT.SMBLDS
INCLUDE       VOLOBJ.SEGMGR.OBJECT.SMBUFF
INCLUDE       VOLOBJ.SEGMGR.OBJECT.SMCHUC
INCLUDE       VOLOBJ.SEGMGR.OBJECT.SMCSGO
INCLUDE       VOLOBJ.SEGMGR.OBJECT.SMDSGB
INCLUDE       VOLOBJ.SEGMGR.OBJECT.SMDSSB
INCLUDE       VOLOBJ.SEGMGR.OBJECT.SMFSID
INCLUDE       VOLOBJ.SEGMGR.OBJECT.SMLOAD
INCLUDE       VOLOBJ.SEGMGR.OBJECT.SMMJCA
INCLUDE       VOLOBJ.SEGMGR.OBJECT.SMMSEG
INCLUDE       VOLOBJ.SEGMGR.OBJECT.SMMTBL
INCLUDE       VOLOBJ.SEGMGR.OBJECT.SMRMVE
INCLUDE       VOLOBJ.SEGMGR.OBJECT.SMSRCH
INCLUDE       VOLOBJ.SEGMGR.OBJECT.SMUNLD
```

END

20.5 LINK CONTROL FILES BUILT DURING SYSTEM GENERATION

Based on the options specified during system generation, a set of link control files is built as SYSLINK, IOULINK, and DMLINK. These files are in the directory .S$SGU$.<system name> for the system you generate. SYSLINK is the link of the major portions of the operating system and varies according to user specification of the following:

* System SVC options

* User SVCs

* File Security

* KIF support

The IOULINK file varies depending on whether or not security is included.

The DMLINK file is the link control file for the disk manager task.

SECTION 21

DNOS SOURCE DISK STRUCTURE

21.1 DIRECTORY STRUCTURE

The following is a directory listing of the directories on a DNOS source disk. Throughout this section, DSC is used as a synonym for the volume on which DNOS source code resides.

```
AGTASK
ALN
ANALZ
ASP
AUI
AXREF
BATCH
BDD
BEMF
BLDPROCS
BMF
CB
CC
CD
CKD
CKR
COM
CONDASM
CONDPASC
CP
CPI
CRV
CSK
CSM
CV
CVD
DCOPY
DD
DEBUG
DEBUGGER
DEVDSR
DIOU
DISKMGR
DNCMS
```

DSCBLD
DXDP
EDITOR
FILEMGR
IBMUTL
IDS
IDT
IFSVC
IOMGR
IOU
IPC
JENED
JOBMGR
KIFMGR
LAGFR
LD
LINK
LINKER
LLR
LOADERS
LOG
LOGON
LS
LSC
LTS
MACROS
MAD
MAILBOX
MCDT
MD
MESSAGES
MKL
MLP
MPC
MPF
MPI
MRF
MS
MSAR
MTE
MVI
NAMMGR
NUCLEUS
O$
OPERATOR
PASASM
PATCH
PERFORM
PF
PICT
PROGMGR
RAL
REQPROC

RESOLVE
RESTART
RVI
RWCRU
S$
S$SGU$
SCI990
SCS
SCU
SD
SDSMAC
SECURITY
SEGMGR
SEM
SIS
SJS
SMMAP
SMS
SND
SOS
SPOOLER
SRFI
SVS
SYSJEN
SYSOVLY
TEMPLATE
TFTPC
TIGRESS
TINFO
TPCALANS
TPDISC
TPLHPC
TPMHPC
UTCOMN
XBJ
XJM
XOI

The following files also appear on the DNOS source disk:

DNOSPROG
TAPEOBJ

21.2 COMPONENTS USED IN BUILDING DNOS

Building DNOS involves creating several batch streams as well as using some batch streams which already exist in the DSC.BATCH.BUILD directory. The batch streams which are created during the build are a product of the Create Batch (CB) utility working with the batch stream templates in the DSC.BATCH.CBINPUT directory. Any process (batch stream) which needs to be performed on a directory of files is generated by CB.

All of the files necessary for building DNOS reside in the DSC.BATCH directory. The components of this directory are described in the paragraphs which follow.

ASSEMBLE
: This is a directory of batch streams, each of which assembles a directory of DNOS source code. In general, a DNOS code directory comprises the modules of a subsystem or single utility. This directory is built using CB and exists only after a DNOS build has been done. Executing the batch stream DSC.BATCH.ASSEMBLE.ALL causes all of the batch streams in this directory to be executed.

BUILD
: This directory consists of the batch streams to build DNOS as well as those used to build the DNOS SCI menus and SCI command procedures. Its files are described further in other portions of this section.

CBINPUT
: When CB is used to create some of the DNOS build batch streams, templates for those batch streams are found in this directory. The templates are described further in other portions of this section.

COMPILE
: Like the ASSEMBLE directory, the COMPILE directory is built using CB. This directory is composed of the batch streams which compile all of the Pascal source in DNOS. The COMPILE directory exists only after a DNOS build has been done. Executing the batch stream DSC.BATCH.COMPILE.ALL causes all of the compilation batch streams in this directory to be executed.

LINK
: This directory contains batch streams to link the whole system. It is built using CB and exists only after a DNOS build has been done. Executing the batch stream DSC.BATCH.LINK.ALL links the whole system.

PICT
: This directory consists of two batch streams, one of which generates PTABLE templates for the ATABLE directory in DSC.TEMPLATE and one of which generates PTABLE templates for the COMMON directory. This directory exists only after a DNOS build has been done.

TRANSLIT

This directory is composed of a set of batch streams which transliterate the source modules of the LINKER directory. It exists only after a DNOS build has been done. Executing the batch stream DSC.BATCH.TRANSLIT.ALL causes all of the transliteration batch streams in this directory to be executed.

The Create Batch (CB) utility processes an input directory, applying to each element of the directory a specified batch stream template. It generates an output batch stream to a file specified when the CB command is issued. The command prompts are documented in the DNOS System Command Interpreter (SCI) Reference Manual.

Files within the DSC.BATCH.CBINPUT directory are used as batch template files for CB when creating many of the DNOS build batch streams. These CBINPUT files are described in the paragraphs which follow.

ALL

This template is used to create a batch stream which executes a directory of batch streams. It is used in the DNOS build process to create the DSC.BATCH.<D>.ALL where <D> is ASSEMBLE, AXREF, COMPILE, DELETE, or PICT, depending on which is needed.

ASSEMBLE

This template is used to create the batch stream to assemble all the modules in one DNOS source directory.

AXREF

This template is used to create a batch stream to produce a cross-reference listing for a given directory of DNOS listings.

CB

This template is used to create the DSC.BATCH.BUILD.<?> batch stream where <?> is one of ASSEMBLE, AXREFS, COMPILE, DELETE, PICT, and TRANSLIT. Each of these batch streams creates the directory of batch streams to process each appropriate directory of DNOS. For example, the batch stream e DSC.BATCH.BUILD.ASSEMBLE built using CB is a batch stream which creates the assembly batch streams (using CB) for each of the source directories of DNOS. This set of assembly batch streams resides in DSC.BATCH.ASSEMBLE. the whole DNOS directory using one of the other CBINPUT templates. For example, it is used to build the directory of assembly batch streams using the ASSEMBLE template.

COMPILE

This template is used to create the batch streams to compile all of the Pascal source modules in one of the DNOS directories.

DELETE

This template is used to create the batch stream to delete the old, outdated object and listing directories for the corresponding DNOS source directory.

FORTRN78

This template is used to create a batch stream to compile all of the FORTRAN source modules in one of the DNOS directories.

LINK

This template is used to build a batch stream to link a part of DNOS.

PICT

This template is used to create the batch stream to build the PTABLE template directory for the appropriate ATABLE or COMMON template directory.

TRANSLIT

This template is used to create the batch stream to transliterate the modules in a given DNOS source directory. The only directory for which this is needed is the LINKER source directory.

The files in the DSC.BATCH.BUILD directory are listed in Table 21-1. The purpose of each file is indicated. Those created during the build process exist only after a DNOS build has been done; these are indicated by a star in the second column of the table. All those not created during a build are further explained in the paragraphs which detail the steps in doing a DNOS build.

Table 21-1 DNOS Batch Stream Files

File Name		Purpose
ASSEMBLE	*	Creates the DSC.BATCH.ASSEMBLE directory of batch streams to assemble all source
AXREFS	*	Creates the DSC.BATCH.AXREF directory of batch streams to generate cross-reference lists
BATCH		Creates batch streams
BSD2		Used for building DNOS from floppy disks
BST2		Used for building DNOS from tape
BST3		Used for building DNOS from tape
BST4		Used for building DNOS from tape
CD1400		Builds a CD1400 system disk
COMPILE	*	Creates the DSC.BATCH.COMPILE directory of batch streams to compile all source
DELETE	*	Creates the DSC.BATCH.DELETE directory of batch streams to delete object and

		listings
DNOS		Builds the DNOS system disk to be shipped
DNOSPROG		Builds a program file with DX10 versions of DNOS utilities needed to build DNOS
DS50		Builds DS50 system disk
DS200		Builds DS200 system disk
DS300		Builds DS300 system disk
DS80		Builds DS80 system disk
LNKDNOS		Builds the linkable parts needed to generate DNOS
MENU		Builds the DNOS SCI menus
MESSAGE1		Builds the source directories and the DNOS usable
MESSAGE2		directories of short and long form messages as well as the batch streams to build DNOS SCI command procedures
OBJECT		Builds the DNOS object from the source by executing the ASSEMBLE, COMPILE, and TRANSLIT batch streams
OBJKIT		Builds a shippable DNOS object disk from DNOS object libraries
PICT	*	Creates the PTABLE templates from the corresponding ATABLE and COMMON templates
PROC0		Creates the privilege level 0 DNOS SCI procedures
PROC2		Creates the privilege level 2 DNOS SCI procedures
PROC4		Creates the privilege level 4 DNOS SCI procedures
PROC6		Creates the privilege level 6 DNOS SCI procedures
PROCSYS		Creates the procedures for .SSYSTEM.S$CMDS
S$LANG		Creates the .S$LANG program file and installs the Assembler and Link Editor
S$OBJECT		Builds the directory .SCI990.S$OBJECT
S$SECURE		Performs the task, procedure, and overlay installations to build the .SSECURE program file.
S$UTIL		Performs the task, procedure, and overlay installations to build the .S$UTIL program file.
SRCKIT		Builds from source kit
TAPE		Builds DNOS onto tape
TRANSLIT	*	Creates the DSC.BATCH.TRANSLIT directory of batch streams to transliterate DNOS source directories

21.3 THE PROCEDURE FOR BUILDING DNOS

The DNOS source disk includes the set of DNOS source modules, a program file of special tasks to build DNOS, and a command procedure library with special commands to build DNOS. The process must be executed using DNOS, following the instructions in the DNOS Source Installation Document.

21.4 DNOS PROGRAM FILES

Table 21-2 and Table 21-3 are maps of the DNOS utility and system program files, respectively. The following flags are used in both tables:

Flags in Program File Maps

FLAG DEFINITIONS

PRI	- PRIORITY	D	- DELETE PROTECTED
S	- SYSTEM	U	- UPDATABLE
P	- PRIVILEGED	O	- OVERFLOW
M	- MEMORY RESIDENT	C	- WRITABLE CONTROL STORE
R	- REPLICATABLE	W	- WRITE PROTECTED
RU	- REUSABLE	SH	- SHARABLE
CP	- COPYABLE	MAP	- RELOCATION BIT MAP PRESENT
E	- EXECUTE PROTECTED	OVLY	- OVERLAY LINK
SP	- SOFTWARE PRIVILEGED	B	- BYPASS SECURITY
P1/S	- PROCEDURE 1 IS ON SAME PROGRAM FILE AS TASK		
P2/S	- PROCEDURE 2 IS ON SAME PROGRAM FILE AS TASK		

Table 21-2 Map of Utility Program File

```
     FILE MAP OF .S$UTIL
       TODAY IS  13:34:26 TUESDAY, SEPTEMBER 27, 1983.

TASK SEGMENTS:  MAX POSSIBLE = 176
ID NAME     LENGTH LOAD PRI S P M R D U RU CP E O C SP B OVLY P1/S P2/S  DATE
01 SCI990   2056   7FA0  1        R                           02/Y 03/Y  8/19/83
02 TINFO    30B8   7FA0  4    P   R                           02/Y 03/Y  8/19/83
03 MS       0C28   1A00  4        R                           02/Y       8/19/83
04 PMTERM   09CA   C000  0  S P                     SP        01/Y       8/19/83
05 IOTBID   0132   C000  0  S P M                   SP        01/Y       8/19/83
06 IPC      1B76   C000  0  S P                     SP        01/Y       8/19/83
07 MAILBOX  24AA   0000  4                                               8/19/83
08 CKD      2096   1A00  4    P   R                 SP        02/Y       8/19/83
09 MPC      1F96   1A00  4        R                           02/Y       8/19/83
0A LOGON    2568   C000  0  S P   R                 SP        01/Y       8/19/83
0B XPD      0DEA   C000  1  S P   R                           01/Y       8/19/83
0C SMM      0BEC   C000  1  S P   R                           01/Y       8/19/83
0D DEBUGGER 70E8   1A00  4    P   R                 SP        02/Y       8/19/83
0E EDITOR   159C   5600  1        R                           02/Y 05/Y  8/19/83
0F TIGR     5194   15A0  4        R                 SP        04/Y       8/19/83
10 MRFSRF   2202   1A00  4        R                 SP        02/Y       8/19/83
11 CCAF     1224   1A00  4        R                           02/Y       8/19/83
12 LS       0C42   1A00  4        R                           02/Y       8/19/83
13 RD       74E0   0000  4    P   R                 SP                   8/30/83
14 VB       6DEC   0000  4    P   R                 SP                   8/30/83
15 CP       4FF0   1A00  4    P   R                           02/Y       8/19/83
16 IOBREAK  023C   C000  0  S P   R     RU                    01/Y       8/19/83
17 SVS      1E28   0000  4    P   R                                      8/19/83
18 RVI      0A48   1A00  4    P   R                 SP        02/Y       8/19/83
19 RESTART  399A   C000  0  S P                               01/Y       8/19/83
1A ANALZ    65B2   1A00  1    P   R                           02/Y       8/19/83
1B IFSVC    2108   1A00  4    P   R                 SP        02/Y       8/19/83
1C XBJS     2454   1A00  4        R                           02/Y       8/19/83
1D MPFMKF   2090   1A00  4        R                           02/Y       8/19/83
1E SCS      24F6   C000  4  S P   R                           01/Y       8/19/83
1F PMSBID   02F4   C000  0  S P                               01/Y       8/19/83
20 IUV      0D58   C000  0  S P                     SP        01/Y       8/19/83
21 LGFORM   3F2A   C000  0  S P         RU                    01/Y       8/19/83
22 DD       1B92   1A00  4        R                           02/Y       8/19/83
23 LD       22EE   1A00  4        R                           02/Y       8/19/83
24 LLR      1D4A   1A00  4        R                           02/Y       8/19/83
25 PMPINS   25C0   C000  0  S P   R                           01/Y       8/19/83
26 MPISPI   1B70   1A00  4        R                 SP        02/Y       8/19/83
27 SMS      0D98   C000  4  S P   R                           01/Y       8/19/83
28 PMPDEL   1CA4   C000  0  S P   R                           01/Y       8/19/83
29 CKR      1580   1A00  4        R                 SP        02/Y       8/19/83
2A IDT      0952   1A00  4                          SP        02/Y       8/19/83
2B SIS      304C   C000  4  S P   R                           01/Y       8/19/83
2C MADSAD   1576   1A00  4    P   R                 SP        02/Y       8/19/83
2D PMRWTK   029C   C000  0  S P         RU                    01/Y       8/19/83
2E SCU      3130   C000  4  S P   R                 SP    49  01/Y       8/19/83
2F SND      0A74   1A00  4        R                           02/Y       8/19/83
```

Table 21-2 Map of Utility Program File (Continued)

30	SJSSTS	2AEE	C000	4	S	P	R				01/Y	8/19/83
31	PMPMAP	0D7A	C000	0	S	P	R	RU			01/Y	8/19/83
32	MD	2C12	1A00	4			R				02/Y	8/19/83
33	SYSGEN	6EC2	0000	4			R			03		8/19/83
34	CD	67C6	0000	4		P	R		SP			8/30/83
35	BD	6258	0000	4		P	R		SP			8/30/83
36	VC	5C68	0000	4		P	R		SP			8/30/83
37	PMPASP	1978	C000	0	S	P	R				01/Y	8/19/83
38	INV	1FFE	C000	0	S	P	R		SP		01/Y	8/19/83
39	AUIDUI	34C0	1A00	4							02/Y	8/19/83
3A	CRV	0D5C	C000	4	S	P	R		SP		01/N	8/19/83
3B	LTS	1794	C000	4	S		R				01/Y	8/19/83
3C	BMF	307E	1A00	4			R				02/Y	8/19/83
3D	LGGLOG	0606	C000	0	S	P					01/Y	8/19/83
3E	MOEMPE	2D24	1A00	4		P	R				02/Y	8/19/83
3F	CPI	1CE6	0000	4		P	R		SP			8/19/83
40	MKL	0FE8	1A00	4		P	R				02/Y	9/13/83
41	CVD	3E08	0000	4		P			SP			8/19/83
42	MVI	1CDA	0000	4		P	R		SP			8/19/83
43	NAMMGR	20D0	C000	0	S	P			SP		01/Y	8/19/83
44	RAL	06C8	C000	4	S	P	R				01/Y	8/19/83
45	CSKCKS	1B9C	1A00	4			R				02/Y	8/19/83
46	RWCRU	0A1E	1A00	4		P	R				02/Y	8/19/83
47	LGACCT	301A	C000	0	S	P		RU			01/Y	8/19/83
48	JOBMGR	26E4	C000	0	S	P		RU	SP		01/Y	8/19/83
49	BEMF	2D92	1A00	4			R				02/Y	8/19/83
4A	PMSBUF	0562	C000	0	S	P					01/Y	8/19/83
4B	IBMUTL	1EF2	0000	4		P	R					8/19/83
4C	RPRCP	254C	C000	0	S	P	R	RU	SP		01/Y	8/19/83
4D	SP$DST	4A0A	11C0	1		P			SP		06/Y	8/19/83
4E	SPINIT	1C22	11C0	1					SP		06/Y	8/19/83
4F	ASP	0D36	1A00	4			R				02/Y	8/19/83
55	SEM	303E	1A00	4			R				02/Y	8/19/83
56	IDS	4A0E	0000	4		P	R		SP			8/19/83
57	PF	227A	1A00	4			R				02/Y	8/19/83
58	MLP	0C3E	C000	4	S		R				01/Y	8/19/83
59	LPWRITER	1F54	4580	4			R	RU			07/Y	8/19/83
5A	LGACHN	01B2	C000	0	S	P	R				01/Y	8/19/83
5B	SPTASK	0A80	0000	4			R					8/19/83
5C	ALN	3940	1A00	4			R				02/Y	8/19/83
5D	DCOPY	41CA	0000	4		P			SP			8/19/83
5E	CSM	3850	0000	4								8/19/83
5F	SCINIT	0CA4	1A00	1			R				02/Y	8/19/83
60	SOS	3332	1A00	4			R				02/Y	8/19/83
61	OPERATOR	3D92	C000	1	S						01/Y	8/19/83
62	XOI	4A5E	1A00	4			R				02/Y	8/19/83
63	LGRCRT	25B6	C000	4	S			RU			01/Y	8/19/83
64	LSC	2F24	1A00	4			R				02/Y	8/19/83
65	CB	2436	1A00	4			R				02/Y	8/19/83
66	SRFI	0FEC	1A00	4		P	R		SP		02/Y	8/19/83
67	DEBUG	0FFA	C000	4	S	P	R				01/Y	8/19/83

Table 21-2 Map of Utility Program File (Continued)

```
68 SAVRES   12B2  C000  0   S P   R                           01/Y       8/19/83
6F TPCALANS 0F72  0000  4         R                                      8/19/83
70 TPDISC   0CF8  0000  4         R                                      8/19/83
71 TPMHPC   0F5A  0000  4         R                                      8/19/83
72 TPLHPC   0EDE  0000  4         R                                      8/19/83
74 XJM      193E  C000  1   S P   R                           01/Y       8/19/83
75 DIOU     0FEC  C000  0   S P   R                 SP        01/Y       8/19/83
83 MCDT     1912  0000  4         R                                      8/19/83
85 BDD      A934  0000  4     P                     SP                   8/19/83
86 CV       5EC0  0000  4     P                     SP                   8/19/83
87 CVINIT   515E  0000  4     P   R                 SP                   8/19/83
88 SD       D800  0000  4     P   R                                      8/19/83
8E RESOLVE  0C7C  1A00  4         R                           02/Y       8/19/83
8F XBJM     03A4  0000  4         R                 SP                   8/19/83
90 RESTART2 2C24  C000  0   S P                               01/Y       8/19/83
93 TFTPC    52C4  0000  4         R                                      8/19/83

PROCEDURE/PROGRAM SEGMENTS: MAX POSSIBLE =  20
ID  NAME   LENGTH LOAD S M R D U SH RU CP E W C OVLY                     DATE
02 S$SYSTEM 19F6  0000           SH         W                            8/19/83
03 SCI990   6586  1A00           SH         W                            8/19/83
04 TIGRESS  159A  0000           SH                                      8/19/83
05 EDITOR   3BEC  1A00           SH                                      8/19/83
06 SPCOMN   11BC  0000           SH       E                              8/19/83
07 LPWRITER 4576  0000           SH         W                            8/19/83

OVERLAYS:  MAX POSSIBLE =  96
ID  NAME   LENGTH LOAD MAP  D  OVLY                                      DATE
01 INIT     1572  21E0                                                   8/19/83
02 INTERACT 4CE2  21E0          01                                       8/19/83
03 BUILD    4C2A  21E0          02                                       8/19/83
04 VERSION  0006  0000 MAP                                               8/19/83
41 SCUINIT  0C4C  E492                                                   8/19/83
42 SCUDEV   0478  E492          41                                       8/19/83
43 SCULDC   05C2  E90A          42                                       8/19/83
44 SCUADD   0434  E492          4E                                       8/19/83
45 SCUPDT   0580  E8C6          44                                       8/19/83
46 SCUDSR   045A  E8C6          4B                                       8/19/83
47 SCUDEL   05B6  E90A          4F                                       8/19/83
48 SCUMISC  0A98  E492          53                                       8/19/83
49 SCUMSP   0B70  E492          48                                       8/19/83
4A SCUAINT  05C4  E8C6          46                                       8/19/83
4B SCUNAME  0462  E8C6          4D                                       8/19/83
4C SCUPD1   040C  E8C6          45                                       8/19/83
4D SCUPD2   055A  E8C6          4C                                       8/19/83
4E SCUMDS   043C  E90A          47                                       8/19/83
4F SCUDATA  0558  E90A          43                                       8/19/83
53 SCUAMUX  02A6  EE8A          54                                       8/19/83
54 SCUAEXP  0248  EE8A          4A                                       8/19/83
```

Table 21-3 Map of System Program File

```
        FILE MAP OF .S$SHIP
          TODAY IS  13:35:05 TUESDAY, SEPTEMBER 27, 1983.

TASK SEGMENTS:  MAX POSSIBLE =   8
ID  NAME     LENGTH LOAD PRI S P M R D U RU CP E O C SP B OVLY P1/S P2/S    DATE
02  PMTBID    0A12  C000  0  S P M                   SP        01/Y        8/19/83
03  IOU       3DAA  C000  0  S P         RU          SP    14  01/Y        8/19/83
04  PMTLDR    18E0  C000  0  S P M                   SP    15  01/Y        8/19/83
05  FILEMGR   380A  C000  0  S P M                   SP    05  01/Y        8/19/83
06  DISKMGR   0730  C000  0  S P M                         11  01/Y        8/19/83
07  PMOVYL    047C  C000  0  S P                     SP        01/Y        8/19/83
08  PMWRIT    02C6  C000  0  S P M                   SP        01/Y        8/19/83

PROCEDURE/PROGRAM SEGMENTS: MAX POSSIBLE =   2
ID  NAME     LENGTH LOAD S M R D U SH RU CP E W C OVLY                      DATE
01  ROOT      3464  0000         U                                         8/19/83
02  S$SHIP    39AA  3480         U                                         8/19/83

OVERLAYS:  MAX POSSIBLE =  69
ID  NAME     LENGTH LOAD MAP  D  OVLY                                       DATE
01  SMTA01    008A  9000                                                   8/19/83
03  SVCSHD    3800  C000                                                   8/19/83
04  SVCTWO    3800  C000                                                   8/19/83
05  KORW      0392  F80A          16                                       8/19/83
06  FORWFB    03F0  F80A                                                   8/19/83
07  FOMISC    0158  F80A          06                                       8/19/83
08  FOXFIL    024C  F80A          07                                       8/19/83
09  FOOPEX    0366  F80A          08                                       8/19/83
0A  KOINSR    03E8  F80A          09                                       8/19/83
0B  KODLSR    02E2  F80A          0A                                       8/19/83
0C  KOOPCL    03D2  F80A          0B                                       8/19/83
0D  KOBDEL    025C  F80A          0C                                       8/19/83
0E  KOBTIS    03A8  F80A          0D                                       8/19/83
0F  KORWS     038E  F80A          0E                                       8/19/83
10  DMOV37    02A6  C48A                                                   8/19/83
11  DMOV38    0280  C48A          10                                       8/19/83
12  CFOVLY    0F44  EE66                                                   8/19/83
13  IUMISCOV 0C18   EE66          12                                       8/19/83
14  IURFAADA 0C74   EE66          13                                       8/19/83
15  PMERRS    0212  D8E0                                                   8/19/83
16  KOPLG     01A6  F80A          0F                                       8/19/83
1B  DSRDSK    0D0C  C000                                                   8/19/83
20  DSR911    1226  C000                                                   8/19/83
3C  JCA000    3000  9000                                                   8/19/83
43  DSR93B    35C6  C000                                                   8/19/83
44  DSR93C    379A  C000                                                   8/19/83
```

SECTION 22

DATA STRUCTURE PICTURES

22.1 OVERVIEW

This section includes details of the templates found in the DSC.TEMPLATE.COMMON directory and in the DSC.TEMPLATE.ATABLE directory. The DSC.TEMPLATE.COMMON directory includes templates for the tables used as assembly language CSEG modules throughout DNOS. It also includes special-purpose templates used only by a single subsystem. The special-purpose templates are not shown in detail here; consult the appropriate TEMPLATE directory for details.

The DSC.TEMPLATE.ATABLE directory contains all of the assembly language versions of the DNOS data structure templates. It includes templates for structures used throughout the operating system as well as templates for special purposes in a single subsystem. This section includes detailed pictures of the general-purpose structures; consult the source directory for details on special-purpose structures.

The template pictures include descriptions of various fields of data structures used by DNOS, their locations, meanings of flags, and special comments. The following features are found in one or more of the structure pictures:

* Header showing the structure name, location in the system, and abbreviation for the name

* Comments describing the use of the structure

* Hexadecimal starting location (or offset relative to the beginning of the structure) for each word of the structure

* Label for each field, chosen from three types:

  - Blank if no label

  - Label of the form FILLxy, if the label is generated by software

  - Label of 6 or fewer characters

- Size of field indicated by space allocated in structure picture

- Comment to right of field, describing that field

- List of flag definitions for each flag field in the structure

  - Flag name

  - Diagram showing position of flag, initial position being 0. The flag is always defined as an assembly language equate for the first bit position shown with an X in the diagram.

  - Description of flag

  - Optional lines of extended explanations of flag settings

- List of equated labels for fields in the structure

  - Label being equated

  - Argument of the equate

  - Value of the equate (or location of the argument)

  - Description of the label being equated

Table 22-1 lists the templates detailed in this section.

Table 22-1 Template Acronyms

Acronym	Meaning
From DSC.TEMPLATE.COMMON	
JMDATA	Job Management Common Area in JCA
LGACOM	Accounting Log Common Data
LGLCOM	System Log Common Data
NFCLKD	Clock Data Area
NFDATA	Global Data Values
NFJOBC	Job Manager Common Area
NFPTR	System Pointers
PMDATA	Global Data Areas for Program Management
From DSC.TEMPLATE.ATABLE	
ACC	Accounting Record Contents
ADR	Alias Descriptor Record
AGR	Access Group Record
BAP	Buffer Address Packet
BRO	Buffered Request Overhead
BTB	B-Tree Block
CCB	Channel Control Block
CDE	Command Definition Entry
CDR	Channel Descriptor Record
CLR	Capabilities List File Record
DDB	DIOU Data Base Definition
DIA	Diagnostic Status
DIT	Disk Information Table
DOR	Directory Overhead Record
DPD	Disk PDT Extension
DPR	Device Utility Parameters
DUS	Device Utility Session Table
FCB	File Control Block - See FSC
FDB	File Directory Block - See FSC
FDP	File Descriptor Packet
FDR	File Descriptor Record
FID	File Identification
FIR	File Information Record
FSC	File Structure Common
FWA	File Manager Work Area
IRB	I/O Request Block
JIT	Job Information Table
JMR	Job Management Request
JSB	Job Status Block
KCB	KIF Currency Block
KDB	Key Descriptor Block (Memory Structure)
KDR	Key Descriptor Block (Disk Structure)
KIB	KIF Information Block
KIT	KIF Task Area

Table 22-1 Template Acronyms (Continued)

Acronym	Meaning
KSB	Keyboard Status Block
LDT	Logical Device Table
LFD	Log File Definition
LPD	Line Printer PDT Extension
LSE	Load Segment Entry
MRB	Master Read/Master Write Buffer
MTX	Extension for a Magnetic Tape
NDB	Name Definition Block
NDS	Name Definition Segment Overhead
NFCRSH	System Crash Code Equates
NFSTAT	Task State Definitions
NRB	Name Manager Request Block
OAD	Overlay Area Description
OAW	Overlay Area Wait Block
OSE	Owned Segment Entry
OTI	Opening Task Identifier
OVB	Overhead Beet
OVT	Overlay Table Entry
PBM	Partial Bit Map Table/Buffer
PDT	Physical Device Table
PFI	Program File Directory Index Entry
PFZ	Program File Record Zero
QHR	Queue Header
QIR	Queued IPC Request
RDB	Request Definition Block
RIB	Return Information Block
RLT	Record Lock Table
ROB	Resource Ownership Block
RPB	Resource Privilege Block
RST	Reserve Segment Table
SAT	Secondary Allocation Table
SCO	Track 0, Sector 0
SDB	Stage Descriptor Block
SGB	Segment Group Block
SLB	System Log Block Formats
SLH	Semaphore List Header
SMR	Segment Manager Request Block
SMT	Segment Manager Table
SOB	Segment Owner Block
SOV	System Overlay Load Table
SSB	Segment Status Block
STA	System Table Area Overhead
STE	Swap Table Entry
SVB	Stage Value Block
TDL	Time Delay List Entry
TSB	Task Status Block

Table 22-1 Template Acronyms (Continued)

Acronym	Meaning
UDO	User Descriptor Overflow Record
UDR	User Descriptor Record
UIP	User ID Parameter
VCB	Value Continuation Block
VDB	Value Definition Block
VRB	Virtual Request Block
XTK	Extension for a Terminal with a Keyboard

Several templates are described in the DNOS SCI and Utilities Design Document, along with detailed descriptions of the utilities that use them. Table 22-2 lists the templates described in that manual.

Table 22-2 Templates Described in SCI and Utilities Document

Acronym	Meaning
ACC	Accounting Record Contents
CNT	Class Name Table
FIR	File Information Record
SCA	System Communications Area
SDEDOR	Memory Resident DOR (UTSORT Structure)
SDEMD	Sorted Directory File Entries Table
SDQ	Spooler Device Queue Entry
SDT	Spooler Device Table Entry
SPM	Spooler Message Format
UDR	User Descriptor Record

22.2 STRUCTURES FROM THE COMMON DIRECTORY

```
************************************************************
*                                                          *
*      JMDATA - JOB MANAGER COMMON AREA        06/08/82    *
*            LOCATED IN JOB MANAGER TASK AREA              *
*                                                          *
************************************************************
*


     *----------+----------*
 >00 !        CURJSB       !   CURRENT JSB POINTER
     +----------+----------+
 >02 !        JMRPTR       !   CURRENT JOB REQUEST
     +----------+----------+
 >04 !        BROPTR       !   CURRENT BRO REQUEST
     +----------+----------+
 >06 !        PARFMT       !   PARENTS FMT POINTER
     +----------+----------+
 >08 !        PARFCB       !   PARENTS FCB POINTER
     +----------+----------+      =============== JOB & TASK STATES =
 >0A !  JSTCRE  !  JSTEXC  !   CREATING
     +----------+----------+      EXECUTABLE
 >0C !  JSTHLT  !  JSTTRM  !   HALTED
     +----------+----------+      TERMINATING
 >0E !  TSTJHT  !  TSTJMR  !   TASK SUSPENDED BY JOBMGR
     +----------+----------+      TASK WAITING ON JMR SVC

+
************************************************************
*                                                          *
*      LGACOM - ACCOUNTING LOG COMMON         04/26/82     *
*               TEMPLATE : LFD                             *
*      (40 MAX = 2 KB OF MESSAGES)                         *
************************************************************


      *----------+----------*
  >00 !        FILL00       !   FLAGS AND ERROR BYTE
      +----------+----------+
  >02 !        FILL01       !   MAX MESSAGE COUNT (0 = NONE)
      +----------+----------+
  >04 !  FILL02  !  FILL03  !   ID OF TASK TO BID ON FULL
      +----------+----------+      ID OF USER TASK TO BID ON FULL
  >06 !        FILL04       !   ACCOUNTING FILE ALLOCATION
      +----------+----------+
  >08 !  FILL05  !          !   LUNOS
      +----------+----------+

  EQUATES:

     LABEL    EQUATE TO    VALUE    DESCRIPTION
     -----    ---------    -----    ------------------------------
     LGACOM   $              >00

+
*****************************************************
*                                                   *
*      LGLCOM - SYSTEM LOG COMMON        09/09/83   *
*              TEMPLATE : LFD                       *
*                                                   *
*****************************************************


      *----------+----------*
  >00 !       FILL00        !   FLAGS (SEE LFD TEMPLATE)
      +----------+----------+
  >02 !       FILL01        !   MAX MESSAGE COUNT (0=>NO MAX)
      +----------+----------+
  >04 !  FILL02  !  FILL03  !   ID OF TASK TO BID ON FULL
      +----------+----------+      ID OF USER TASK TO BID ON FULL
  >06 !  FILL04  !          !   LOG DEVICE NAME (BLANKS=>NONE)
      +----------+----------+
  >08 !          !          !
      +----------+----------+
  >0A !  FILL05  !          !   FILENAME 1
      +----------+----------+
      /          /          /
      /          /          /
      +----------+----------+
  >12 !  FILL06  !          !   FILENAME 2
      +----------+----------+
      /          /          /
      /          /          /
      +----------+----------+
  >1A !       FILL07        !   LOG FILE ALLOCATION
      +----------+----------+
  >1C !  FILL08  !          !   LUNOS
      +----------+----------+

  EQUATES:

     LABEL   EQUATE TO   VALUE   DESCRIPTION
     -----   ---------   -----   ------------------------------
     LGLCOM  $             >00

+
*************************************************************
*                                                           *
*      NFCLKD - CLOCK DATA AREA                 01/31/82    *
*                                                           *
*************************************************************
* THIS COMMON SEGMENT INCLUDES FLAGS AND COUNTERS USED FOR
* PERFORMANCE DATA GATHERING AND SYSTEM CLOCK WORKSPACES.
* THE WORKSPACE STARTING AT CLKWP2 IS USED FOR UPDATING THE
* CLOCK; THAT AT CLKWP IS THE NORMAL CLOCK WORKSPACE.  IN
* THE LATTER, THE ESTIMATED UTILIZATION VARIABLES (R4,R5)
* CONTAIN VALUES IN THE RANGE 0 THROUGH >8000 WHERE 0
* REPRESENTS 0% UTILIZATION AND >8000 REPRESENTS 100%
* UTILIZATION.
*


      *----------+----------*
  >00 !       NBFLGS        !   NUMBER OF STATISTIC FLAGS
      +----------+----------+
  >02 !       NBSAM1        !   NUMBER SAMPLES ON FLAGS (WD 1)
      +----------+----------+
  >04 !       NBSAM2        !   NUMBER SAMPLES ON FLAGS (WD 2)
      +----------+----------+
  >06 !       STFLG0        !   FLAG 0 - FOR DISK UTILITY
      +----------+----------+
  >08 !       FLG0H1        !   HIT COUNT FOR FLAG 0 (WD 1)
      +----------+----------+
  >0A !       FLG0H2        !   HIT COUNT FOR FLAG 0 (WD 2)
      +----------+----------+
  >0C !       STFLG1        !   FLAG 1 - CPU UTILIZATION
      +----------+----------+
  >0E !       FLG1H1        !   HIT COUNT FOR FLAG 1 (WD 1)
      +----------+----------+
  >10 !       FLG1H2        !   HIT COUNT FOR FLAG 1 (WD 2)
      +----------+----------+
  >12 !       STFLG2        !   FLAG 2 - SCHEDULER
      +----------+----------+
  >14 !       FLG2H1        !   HIT COUNT FOR FLAG 2 (WD 1)
      +----------+----------+
  >16 !       FLG2H2        !   HIT COUNT FOR FLAG 2 (WD 2)
      +----------+----------+
  >18 !       STFLG3        !   FLAG 3 - FILE MANAGER
      +----------+----------+
  >1A !       FLG3H1        !   HIT COUNT FOR FLAG 3 (WD 1)
      +----------+----------+
  >1C !       FLG3H2        !   HIT COUNT FOR FLAG 3 (WD 2)
      +----------+----------+
  >1E !       STFLG4        !   FLAG 4 - TASK LOADER
      +----------+----------+
  >20 !       FLG4H1        !   HIT COUNT FOR FLAG 4 (WD 1)
      +----------+----------+
  >22 !       FLG4H2        !   HIT COUNT FOR FLAG 4 (WD 2)
```

```
     +----------+----------+
>24  !       STFLG5        !   FLAG 5 - MAP ONE ACTIVITY
     +----------+----------+
>26  !       FLG5H1        !   HIT COUNT FOR FLAG 5 (WD 1)
     +----------+----------+
>28  !       FLG5H2        !   HIT COUNT FOR FLAG 5 (WD 2)
     +----------+----------+
>2A  !       STFLG6        !   FLAG 6 - SVC CODE FILE MGR
     +----------+----------+
>2C  !       FLG6H1        !   HIT COUNT FOR FLAG 6 (WD 1)
     +----------+----------+
>2E  !       FLG6H2        !   HIT COUNT FOR FLAG 6 (WD 2)
     +----------+----------+
>30  !       STFLG7        !   FLAG 7
     +----------+----------+
>32  !       FLG7H1        !   HIT COUNT FOR FLAG 7 (WD 1)
     +----------+----------+
>34  !       FLG7H2        !   HIT COUNT FOR FLAG 7 (WD 2)
     +----------+----------+
>36  !       STFLG8        !   FLAG 8
     +----------+----------+
>38  !       FLG8H1        !   HIT COUNT FOR FLAG 8 (WD 1)
     +----------+----------+
>3A  !       FLG8H2        !   HIT COUNT FOR FLAG 8 (WD 2)
     +----------+----------+
>3C  !       STFLG9        !   FLAG 9
     +----------+----------+
>3E  !       FLG9H1        !   HIT COUNT FOR FLAG 9 (WD 1)
     +----------+----------+
>40  !       FLG9H2        !   HIT COUNT FOR FLAG 9 (WD 2)
     +----------+----------+
>42  !       STFLGA        !   FLAG 10
     +----------+----------+
>44  !       FLGAH1        !   HIT COUNT FOR FLAG 10 (WD 1)
     +----------+----------+
>46  !       FLGAH2        !   HIT COUNT FOR FLAG 10 (WD 2)
     +----------+----------+
>48  !       STFLGB        !   FLAG 11
     +----------+----------+
>4A  !       FLGBH1        !   HIT COUNT FOR FLAG 11 (WD 1)
     +----------+----------+
>4C  !       FLGBH2        !   HIT COUNT FOR FLAG 11 (WD 2)
     +----------+----------+
>4E  !       STCNT0        !   COUNTER 0 - # JOBS COMPLETED
     +----------+----------+
>50  !       STCNT1        !   COUNTER 1 - # TASKS COMPLETED
     +----------+----------+
>52  !       STCNT2        !   COUNTER 2 - # SEG MGR CALLS
     +----------+----------+
>54  !       STCNT3        !   COUNTER 3 - # FILE MGR CALLS
     +----------+----------+
>56  !       STCNT4        !   COUNTER 4 - # IPC CALLS
     +----------+----------+
>58 !        STCNT5        !   COUNTER 5 - # ROLL OUTS
    +----------+-----------+
>5A !        STCNT6        !   COUNTER 6 - # FILE MGR Q REQ
    +----------+-----------+
>5C !        STCNT7        !   COUNTER 7 - # SYSTEM OVLY LDS
    +----------+-----------+
>5E !        STCNT8        !   COUNTER 8 - # NAME MANAGER CALLS
    +----------+-----------+
>60 !        STCNT9        !   COUNTER 9 - # IOU CALLS
    +----------+-----------+
>62 !        STCNTA        !   COUNTER 10- # SYSTAB SCHED CALLS
    +----------+-----------+
>64 !        STCNTB        !   COUNTER 11
    +----------+-----------+
>66 !        CLKWP2        !   R0
    +----------+-----------+
>68 !        CKTIC1        !   R1  - 32 BIT CLOCK TIC COUNTER
    +----------+-----------+
>6A !        CKTIC2        !   R2  -    WORDS 1 AND 2
    +----------+-----------+
>6C !        YEAR          !   R3  - CLOCK YEAR COUNTER
    +----------+-----------+
>6E !        DAY           !   R4  - CLOCK DAY COUNTER
    +----------+-----------+
>70 !        HOUR          !   R5  - CLOCK HOUR COUNTER
    +----------+-----------+
>72 !        MIN           !   R6  - CLOCK MINUTE COUNTER
    +----------+-----------+
>74 !        SEC           !   R7  - CLOCK SECOND COUNTER
    +----------+-----------+
>76 !        TIC           !   R8  - CLOCK TIC COUNTER
    +----------+-----------+
>78 !        FILL00        !   R9  - SECONDS PER MINUTE
    +----------+-----------+
>7A !        FILL01        !   R10 - HOURS PER DAY
    +----------+-----------+
>7C !        FILL02        !   R11 - DAYS PER YEAR
    +----------+-----------+
>7E !        FILL03        !   R12 - SCRATCH
    +----------+-----------+
>80 !        FILL04        !   R13 - SCRATCH
    +----------+-----------+
>82 !        FILL05        !   R14
    +----------+-----------+
>84 !        FILL06        !   R15 - TIC COUNT FOR TIME UNITS
    +----------+-----------+
>86 !        CLKWP         !   R0  - SCRATCH
    +----------+-----------+
>88 !        FILL07        !   R1  - SCRATCH
    +----------+-----------+
>8A !        FILL08        !   R2  - SCRATCH
    +----------+-----------+
>8C !        FILL09        !   R3  - FOR BAR GRAPH
```

```
    +----------+----------+
>8E !        DSUTIL       !   R4  - ESTIMATED DISK UTILIZ
    +----------+----------+
>90 !        CPUTIL       !   R5  - ESTIMATED CPU UTILIZ
    +----------+----------+
>92 !        TSTIC        !   R6  - TIME SLICE TIC COUNT
    +----------+----------+
>94 !        DSPFG1       !   R7  - INDEX TO FLAGS TO DISPLAY
    +----------+----------+
>96 !        DSPFG2       !   R8  -      ON FRONT PANEL
    +----------+----------+
>98 !        FILL0A       !   R9  - FOR FRONT PANEL DISPLAY
    +----------+----------+
>9A !        FILL0B       !   R10 - FOR FRONT PANEL DISPLAY
    +----------+----------+
>9C !        FILL0C       !   R11 - SCRATCH
    +----------+----------+
>9E !        FILL0D       !   R12 - FRONT PANAL ADDRESS
    +----------+----------+
>A0 !        FILL0E       !   R13 - WORKSPACE POINTER
    +----------+----------+
>A2 !        FILL0F       !   R14 - PROGRAM COUNTER
    +----------+----------+
>A4 !        FILL10       !   R15 - STATUS
    +----------+----------+
>A6 !        PFG0H2       !   PREV NUMBER HITS FLAG 0 (WD 2)
    +----------+----------+
>A8 !        PFG1H2       !   PREV NUMBER HITS FLAG 1 (WD 2)
    +----------+----------+
```

EQUATES:

LABEL	EQUATE TO	VALUE	DESCRIPTION
SFESIZ	6	>06	STATISTICS FLAGS ENTRY SIZE
NFCSIZ	$-NBFLGS	>AA	

```
***********************************************************
*                                                         *
*      NFDATA - GLOBAL DATA VALUES             03/08/83   *
*                                                         *
***********************************************************
* THIS COMMON SEGMENT CONTAINS GLOBAL DATA VALUES INCLUDING
* THE FOLLOWING:  BEET ANCHORS FOR THE TIME ORDERED LIST
* (TOL), CACHE LIST, FREE MEMORY LIST, STATIC BUFFER AREA,
* AND THE TEMPORARY MEMORY BUFFER; PARAMETERS FOR PRIORITY
* COMPUTATION AND SCHEDULING; ROLL OUT AND LOAD PARAMETERS.
* A NUMBER OF MISCELLANEOUS DATA VALUES ARE ALSO FOUND IN
* THIS CSEG.  COMMENTS IN THIS TEMPLATE'S SOURCE FILE SHOW
* PRIORITY, AGING, AND ROLLOUT PARAMETERS.
*
* NOTE: CHANGES TO THIS TEMPLATE REQUIRE CORRESPONDING
*       CHANGES TO SYSGEN.
*


      *----------+----------*
  >00 !        TMTOL        !   START OF TIME ORDERED LIST
      +----------+----------+
  >02 !        TOLBET       !   BEET ADDRESS OF TOL HEADER
      +----------+----------+
  >04 !        TMTOLN       !   FORWARD POINTER
      +----------+----------+
  >06 !        TMTOLO       !   BACKWARD POINTER
      +----------+----------+
  >08 !        TMTTYP       !   TYPE OF BLOCK
      +----------+----------+
  >0A !        FILL00       !
      +----------+----------+
  >0C !        FILL01       !   SCHEDULER ENTRY VECTOR (WP)
      +----------+----------+
  >0E !        FILL02       !   (PC)
      +----------+----------+
  >10 !        FILL03       !   (ST)
      +----------+----------+
  >12 !  FILL04  !          !   RESERVED
      +----------+----------+
  >14 !          !          !
      +----------+----------+
  >16 !        INTCDT       !   FAKE CDT FOR SYSTEM INIT. TASK
      +----------+----------+
  >18 !  RESPFL  !  RESTSK  !   PROGRAM FILE LUNO
      +----------+----------+      ID OF SYSTEM RESTART TASK
  >1A !        RESTRT       !   ID OF USER RESTART TASK
      +----------+----------+
  >1C !        FILL05       !
      +----------+----------+
  >1E !        RELOCA       !   RELOCATION VALUE FOR LOADER
      +----------+----------+
>20 !        CHELST         !   START OF CACHE LIST
    +----------+----------+
>22 !        CHEBET         !   BEET ADDRESS OF LIST HEADER
    +----------+----------+
>24 !        CHEFWD         !   FORWARD POINTER
    +----------+----------+
>26 !        CHEBKW         !   BACKWARD POINTER
    +----------+----------+
>28 !        CHETYP         !   TYPE OF BLOCK
    +----------+----------+
>2A !        SMTBMP         !   SEG MANAGER SCRATCH WORD
    +----------+----------+
>2C !  USERPF  !          !   NAME OF PF CONTAINING USER-
    +----------+----------+
    /          /          /
    /          /          /
    +----------+----------+
>34 !        TSKDOA         !   FOR INTERRUPT 2 PROCESSOR TO
    +----------+----------+      RETURN TASK ERR CODE TO SCHD.
>36 !        FILL06         !   * RESERVED * (SMRID)
    +----------+----------+
>38 !        CPU12          !   SET IF 990/12 CPU
    +----------+----------+
>3A !        AJSBCT         !   ACTIVE JSB COUNT
    +----------+----------+
>3C !        ATSBCT         !   ACTIVE TSB COUNT
    +----------+----------+
>3E !        WTSBCT         !   COUNT OF TSB'S ON WOM
    +----------+----------+
>40 !        UAHEAD         !   START OF FREE USER AREA
    +----------+----------+
>42 !        UAPTR          !   BEET ADDRESS OF BLOCK
    +----------+----------+
>44 !        UAFWD          !   FOWARD LINK POINTER
    +----------+----------+
>46 !        UABKW          !   BACKWARD LINK POINTER
    +----------+----------+
>48 !        UABADD         !   START ADDRESS OF USER MEMORY
    +----------+----------+
>4A !        UATLEN         !   TOTAL LENGTH OF USER MEMORY
    +----------+----------+
>4C !        UADSTR         !   START OF DYNAMIC USER MEMORY
    +----------+----------+
>4E !        UADLEN         !   LENGTH OF DYNAMIC USER MEMORY
    +----------+----------+
>50 !        UADMIN         !   MIN AMOUNT OF DYNAMIC MEMORY
    +----------+----------+
>52 !        USEMEM         !   SUM OF ALL CURRENT FREE MEMORY
    +----------+----------+
>54 !        USEFRG         !   NUMBER OF FREE MEMORY FRAGMENT
    +----------+----------+
>56 !        TICFRQ         !   CLOCK FREQUENCY (TICS/SEC)
    +----------+----------+
>58 !        UNTSLC         !   CLOCK TICS PER TIME SLICE
    +-----------+-----------+
>5A !         TPU           !   TICS PER TIME UNIT
    +-----------+-----------+
>5C !        TSENAB         !   TIME SLICE ENABLE FLAG
    +-----------+-----------+
>5E !        TICLMT         !   LIMIT FOR CURRENT TIME SLICE
    +-----------+-----------+
>60 !        BTAHED         !   SIZE OF ANCHOR BLOCK
    +-----------+-----------+
>62 !        BTAPTR         !   BEET ADDRESS OF THIS BLOCK
    +-----------+-----------+
>64 !        BTAFWD         !   FORWARD POINTER
    +-----------+-----------+
>66 !        BTAREV         !   REVERSE POINTER
    +-----------+-----------+
>68 !        BTAADD         !   BEET ADDRESS OF TABLE AREA
    +-----------+-----------+
>6A !        BTALEN         !   LENGTH OF TABLE AREA IN BEETS
    +-----------+-----------+
>6C !        BTAMAX         !   MAXIMUM AREA FOR BUFFERS
    +-----------+-----------+
>6E !        BTAALL         !   ALLOCATED TABLE AREA
    +-----------+-----------+
>70 !        BTAHDN         !   HIDDEN TABLE AREA
    +-----------+-----------+
>72 !  FILL07   !           !   RESERVED
    +-----------+-----------+
>74 !        MEMSIZ         !   SIZE OF SYSTEM IN BEETS
    +-----------+-----------+
>76 !        CMEMSZ         !   SIZE OF CRASH FILE IN BEETS
    +-----------+-----------+
>78 !        CRSHTL         !   CRASH FILE TILINE ADDRESS
    +-----------+-----------+
>7A !  CRSHHD   !  CRSHSC   !   CRASH FILE HEAD ADDRESS
    +-----------+-----------+      CRASH FILE SECTOR ADDRESS
>7C !        CRSHCL         !   CRASH FILE CYLINDER ADDRESS
    +-----------+-----------+
>7E !        CRSHSL         !   CRASH FILE TILINE SELECT
    +-----------+-----------+
>80 !        TMBHED         !   SIZE OF ANCHOR BLOCK
    +-----------+-----------+
>82 !        TMBPTR         !   BEET ADDRESS OF THIS BLOCK
    +-----------+-----------+
>84 !        TMBFWD         !   FORWARD POINTER
    +-----------+-----------+
>86 !        TMBBWD         !   BACK POINTER
    +-----------+-----------+
>88 !        TMBADD         !   TEMP ADDRESS BOUNDARY
    +-----------+-----------+
>8A !        TMBLEN         !   TEMP BUFFER LENGTH
    +-----------+-----------+
>8C !        FILL08         !   RESERVED
```

```
      +----------+----------+
 >8E  !        FILL09       !   RESERVED
      +----------+----------+
 >90  !        EXTIME       !   EXTEND TIME SLICE FLAG
      +----------+----------+
 >92  !        FUTPDT       !   IOU PDT CURRENTLY IN USE
      +----------+----------+
 >94  !        UNLPDT       !   UNLOAD VOLUME PDT IN USE
      +----------+----------+
 >96  !        SYSUNT       !   ELASPED SYSTEM TIME UNITS
      +----------+----------+
 >98  !        JCABT        !   BEET ADDRESS OF JCA
      +----------+----------+
 >9A  !        TDLEXP       !   TIME DELAY EXPIRED FLAG
      +----------+----------+
 >9C  !        WJSBCT       !   WOM LIST JSB COUNT
      +----------+----------+
 >9E  !  LDTDSC  !          !
      +----------+----------+
      /          /          /
      /          /          /
      +----------+----------+
 >AC  !        IOINDX       !   VALUE OF X IN I/O INDICATOR
      +----------+----------+      FORMULA.
 >AE  !  INTPRI  !          !   INSTALLED PRI 1 -> 188
      +----------+----------+
 >B0  !          !          !
      +----------+----------+
 >B2  !  JPRMOD  !          !   VALUE FOR INSTALLED PRI OF 1
      +----------+----------+
 >B4  !          !          !
      +----------+----------+
 >B6  !  DYNMOD  !          !   VALUE FOR INSTALLED PRI OF 1
      +----------+----------+
 >B8  !          !          !
      +----------+----------+
 >BA  !  AGEIND  !          !   VALUE FOR INSTALLED PRI OF 1
      +----------+----------+
 >BC  !          !          !
      +----------+----------+
 >BE  !        ENDLMT       !   END ACTION EXECUTION TIME LIMIT
      +----------+----------+      IN SYSTEM TIME UNITS
 >C0  !        CLMXBF       !   MAX # BUFFERS ON CACHE LIST
      +----------+----------+
 >C2  !        CLMXPS       !   MAX # PROGRAM SEGS ON CACHE LIST
      +----------+----------+
 >C4  !        CLMNBF       !   RESERVED
      +----------+----------+      WAS MIN # BUFFERS ON CACHE LIST
 >C6  !        CLNBUF       !   # BUFFERS CURRENTLY ON CACHE LIST
      +----------+----------+
 >C8  !        CLNPRG       !   # PROG SEGS CURRENTLY ON CACHE LIST
      +----------+----------+      ------
 >CA  !        TLSPND       !   MIN # SYS TIME UNITS TASK MUST
```

```
       +----------+----------+      BE SUSPENDED BEFORE ELIGIBLE FOR
   >CC !       TLEXEC        !   MIN # SYS TIME UNITS OF EXECUTION
       +----------+----------+      TASK MUST RECEIVE BEFORE
   >CE !       TOLCNT        !   # TASKS ON TOL ELIGIBLE FOR ROLL
       +----------+----------+      OUT (NOT MEMORY RESIDENT)
   >D0 !       TOLS24        !   IF NOT 0 STATE 24 TASKS ARE
       +----------+----------+      IMMEDIATELY ELIGIBLE FOR
   >D2 !       LDRTDY        !   TASK LOADER TIME DELAY VALUE
       +----------+----------+      IN SYSTEM TIME UNITS
   >D4 !       NUMROL        !   NUMBER OF SEGMENTS ROLLED OUT
       +----------+----------+
   >D6 !       ROLSPA        !   AMOUNT OF ROLL SPACE USED
       +----------+----------+
   >D8 !       LDREXC        !   TASK LOADER IS EXECUTING FLAG
       +----------+----------+
   >DA !       TSKCNT        !   COUNT OF TASKS IN SYSTEM
       +----------+----------+
   >DC !       FRCROL        !   FORCED ROLL-OUT COUNT
       +----------+----------+
   >DE !       PMSTSB        !   ADDRESS OF TSB TO ROLL
       +----------+----------+
   >E0 !  SITENM  !          !   SITE NAME
       +----------+----------+
       /          /          /
       /          /          /
       +----------+----------+
   >E8 !  MGRCG   !          !   SYSTEM MANAGER CONTROL GROUP
       +----------+----------+
       /          /          /
       /          /          /
       +----------+----------+
   >F0 !  PUBLIC  !          !   PUBLIC ACCESS GROUP
       +----------+----------+
       /          /          /
       /          /          /
       +----------+----------+
   >F8 !       SYSOPT        !   SYSGEN OPTIONS WORD(FLAGS BELOW)
       +----------+----------+
   >FA !       JCARES        !   RESERVED TABLE AREA AMOUNT
       +----------+----------+
   >FC !       EXPLEN        !   LENGTH TO EXPAND TABLE AREA
       +----------+----------+
   >FE !       CONTRY        !   COUNTRY CODE FOR THIS SYSTEM
       +----------+----------+
 >0100 !       ITSKMX        !   MAX ALLOC FOR GET AND PUT DATA
       +----------+----------+
 >0102 !       ITSKCR        !   CURR ALLOC FOR GET & PUT DATA
       +----------+----------+
 >0104 !       DCPYAC        !   DCOPY ACTIVE INDICATOR
       +----------+----------+      0=DCOPY NOT ACTIVE, 1=ACTIVE
 >0106 !  VERS    !          !   VERSION NUMBER
       +----------+----------+
 >0108 !          !          !
```

```
      +----------+----------+
>010A !          !          !
      +----------+----------+
>010C !        MEMTIC       !   COUNT BEFORE MEM CNTRL CHECK
      +----------+----------+
>010E !  SYSTEM  !          !   NAME OF SYSTEM
      +----------+----------+
      /          /          /
      /          /          /
      +----------+----------+
>0116 !        COMFLG       !   TO SCHEDULE OR NOT TO SCHEDULE
      +----------+----------+
>0118 !        WCSMAP       !   LD MAP TO LOAD WCS (LIMIT)
      +----------+----------+
>011A !        FILLOB       !   (BIAS)
      +----------+----------+
>011C !        FILLOC       !   RESERVED
      +----------+----------+
>011E !        IOTFLG       !   SCHEDULER, IOTBID FLAG
      +----------+----------+      0=BID REQ OUTSTANDING->BID NFTBID
>0120 !        CLOCNT       !   FILE CLOSES OUTSTANDING
      +----------+----------+
>0122 !        CPUID        !   CPU ID
      +----------+----------+
>0124 !  ATTNDV  !          !   ATTENTION DEVICE NAME
      +----------+----------+
>0126 !          !          !
      +----------+----------+
>0128 !  FILLOD  !          !   RESERVED
      +----------+----------+
>012A !          !          !
      +----------+----------+
>012C !  CLFLUN  !  FILLOE  !   STORAGE PLACE FOR LUNO TO .S$CLF
      +----------+----------+      RSVD-FORCE CLFLUN IN LEFT BYTE
>012E !        UALGFB       !   LARGEST FREE BLOCK OF DYNAMIC MEM
      +----------+----------+
>0130 !        TILADD       !   SAVE TILINE ADDR FOR POWER UP-MUX
      +----------+----------+
>0132 !        PWRFLG       !   CONTROLLER POWERUP FLAG-MUX
      +----------+----------+
```

```
  FLAGS FOR FIELD: SYSOPT    #F8 - SYSGEN OPTIONS WORD(FLAGS BELOW)

    OPTDSK = (X...............) - SYSTEM DISK PRESENT
    OPTMFM = (.X..............) - MINIMUM FILE MANAGEMENT PRESENT
    OPTCDF = (..X.............) - CREATE/DELETE FILE CAPABILITY
    OPTBLK = (...X............) - BLOCKED FILE CAPABILITY
    OPTBSF = (....X...........) - BLANK SUPPRESSED FILE CAPABILITY
    OPTEXF = (.....X..........) - FILE EXTENSION CAPABILITY
    OPTACC = (......X.........) - ACCOUNTING DATA COLLECTED
    OPTOSP = (.......X........) - OUTPUT SPOOLING
    OPTIPC = (........X.......) - IPC PRESENT
    OPTSEC = (.........X......) - SECURITY
    OPTKIF = (..........X.....) - KIF PRESENT
    OPTEXJ = (...........X....) - EXPANDABLE JCA
    OPTRAW = (............X...) - DM READ AFTER WRITE ENABLED
    OPTWCS = (.............X..) - 1=PERFORMANCE WCS
    OPTPFR = (..............X.) - 1=POWER FAIL RECOVERY
           = (...............X) - RESERVED
```

EQUATES:

LABEL	EQUATE TO	VALUE	DESCRIPTION
NFSBWP	$-26	>FFF2	
NFSIZE	$-TMTOL	>134	SIZE OF THIS CSEG

```
************************************************************
*                                                          *
*      NFJOBC - JOB MANAGER COMMON AREA          11/09/79 *
*                                                          *
************************************************************
* THIS COMMON SEGMENT CONTAINS DATA VALUES USED BY THE
* JOB MANAGER.


      *----------+----------*
  >00 !        NXTJID       !   NEXT AVAILABLE JOB ID
      +----------+----------+
  >02 !        FSTJID       !   BEGINING AVAILABLE JOB ID
      +----------+----------+
  >04 !        LSTJID       !   LAST AVALIBLE ID IN LIST
      +----------+----------+
  >06 !        JOBCNT       !   NUMBER OF JOBS IN SYSTEM
      +----------+----------+
  >08 !        JOBLMT       !   SYSTEM LIMIT ON ACTIVE JOBS
      +----------+----------+
  >0A !        JCAMIN       !   SIZE OF JCA IN BYTES (SMALL)
      +----------+----------+
  >0C !        JCAAVG       !   SIZE OF JCA IN BYTES (MED)
      +----------+----------+
  >0E !        JCAMAX       !   SIZE OF JCA IN BYTES (LARGE)
      +----------+----------+
  >10 !        JWTQUE       !   FOREGROUND JOB WAIT LIST
      +----------+----------+
  >12 !        JOBQ         !   JOB MANAGER REQUEST QUEUE
      +----------+----------+
  >14 !        JOBBCT       !   BACKGROUND JOB COUNT
      +----------+----------+
  >16 !        JOBBLM       !   BACKGROUND JOB LIMIT
      +----------+----------+
  >18 !        JOBBWT       !   BACKGROUND JOB WAIT LIST
      +----------+----------+


  EQUATES:

     LABEL   EQUATE TO   VALUE   DESCRIPTION
     -----   ---------   -----   ------------------------------
     NFJSIZ   $-NXTJID     >1A     CSEG SIZE

+
***********************************************************
*                                                         *
*      NFPTR  - SYSTEM POINTERS                09/20/83   *
*                                                         *
***********************************************************
* THIS COMMON SEGMENT CONTAINS POINTERS USED BY MANY PARTS
* OF DNOS.


      *----------+----------*
  >00 !        TDLHDR       !   TIME DELAY LIST HEADER
      +----------+----------+
  >02 !        WOMQUE       !   WAITING ON MEMORY QUEUE HEADER
      +----------+----------+
  >04 !        EXTSB        !   CURRENTLY EXECUTING TASK
      +----------+----------+
  >06 !        EXJSB        !   EXECUTING TASK JSB ADDRESS
      +----------+----------+
  >08 !        PDTLST       !   START OF PDT LIST
      +----------+----------+
  >0A !        LDTLST       !   START OF LDT LIST
      +----------+----------+
  >0C !        JSBLST       !   START OF JSB LIST
      +----------+----------+
  >0E !        ACTJSB       !   START OF ACTIVE JSB LIST
      +----------+----------+
  >10 !        WOMJSB       !   START OF JSBS WAITING ON MEMORY
      +----------+----------+
  >12 !        JCASTR       !   START OF ALL JCA AREAS
      +----------+----------+
  >14 !        MBSSTR       !   POINTER TO SYSTEM SCB
      +----------+----------+
  >16 !        PDTSAV       !   POINTER TO SAVED PDT FOR DSRS
      +----------+----------+
  >18 !        MAPSHD       !   POINTER TO SCHEDULER MAP FILE
      +----------+----------+
  >1A !        MAPSV2       !   POINTER TO SVC SECOND MAP FILE
      +----------+----------+
  >1C !        CURMAP       !   POINTER TO CURRENT MAP 0 FILE
      +----------+----------+
  >1E !        RUTSSB       !   POINTER TO SSB FOR ROOT
      +----------+----------+
  >20 !        COMSSB       !   SSB ADDR OF SYSTEM COMMON
      +----------+----------+
  >22 !        SMSTR        !   SSB ADDR OF FIRST SM SEGMENT
      +----------+----------+
  >24 !        SMEND        !   SSB ADDR OF LAST SM SEGMENT
      +----------+----------+
  >26 !        FMSTR        !   SSB ADDR OF FIRST SM SEGMENT
      +----------+----------+
  >28 !        FMEND        !   SSB ADDR OF LAST SM SEGMENT
      +----------+----------+
   >2A !         YRPTR          !    PTR TO YEAR COUNTER (DATE&TIME)
       +-----------+-----------+
             (CONTINUED)

+
       +-----------+-----------+
   >2C !         SYSTAB         !    OVERHEAD PTR FOR TABLE AREA
       +-----------+-----------+
   >2E !         SPATCH         !    START OF PATCH AREA S$$PAT
       +-----------+-----------+
   >30 !         IXPTR          !    ILLEGAL PC
       +-----------+-----------+
   >32 !         PCAPTR         !    MUX DEV/INT ENTRY
       +-----------+-----------+
   >34 !         DTMRAD         !    ADDRESS OF IODTMR
       +-----------+-----------+
   >36 !         FILL00         !    RESERVED
       +-----------+-----------+
   >38 !         FILL01         !    RESERVED
       +-----------+-----------+
   >3A !         FILL02         !    RESERVED
       +-----------+-----------+
   >3C !         FILL03         !    RESERVED
       +-----------+-----------+
   >3E !         TRCPTR         !    POINTER TO /12 TRACE SAVE AREA
       +-----------+-----------+
   >40 !         BIDREQ         !    ANCHOR FOR BID REQUESTS
       +-----------+-----------+
   >42 !         EORNKR         !    ANCHOR FOR EOR REQUESTS
       +-----------+-----------+
   >44 !         SYSJSB         !    POINTER TO SYSTEM JOB JSB
       +-----------+-----------+
   >46 !         CCBSTR         !    START OF THE GLOBAL CCBS
       +-----------+-----------+
   >48 !   SCOTID  !            !    NFTBID TASK ID AND LUNO
       +-----------+-----------+
   >4A !         MAP1SV         !    POINTER TO MAP FILE 1 SAVE
       +-----------+-----------+       AREA FOR LEVEL 2 INTERRUPTS
   >4C !         MSGQUE         !    PTR TO PUT DATA MESSAGES
       +-----------+-----------+
   >4E !         SOPJSB         !    SYSTEM OPERATOR JSB ADDRESS
       +-----------+-----------+
   >50 !         SYSPDT         !    SYSTEM DISK PDT ADDRESS
       +-----------+-----------+
   >52 !         ETBPTR         !    EXPANSION CHASSIS TABLE
       +-----------+-----------+
   >54 !         PCSPTR         !    SINGLE DEV/INT ENTRY
       +-----------+-----------+
   >56 !         PCMPTR         !    MULTIPLE DEV/INT ENTRY
       +-----------+-----------+
   >58 !         PCEPTR         !    EXPANSION CHASSIS ENTRY
       +-----------+-----------+
   >5A !         OVYTAB         !    SYSTEM OVERLAY TABLE ADDRESS
```

```
       +----------+----------+
  >5C  !       IDTAB         !   IPL LOADED OVERLAY TABLE ADDR.
       +----------+----------+
  >5E  !       COMPTR        !   POINTER TO COMM MODULE
       +----------+----------+
  >60  !       BADWP         !   ILLEGAL XOP WORKSPACE
       +----------+----------+
  >62  !       CLOK12        !   12 MS CLOCK VECTOR
       +----------+----------+
             (CONTINUED)

+
       +----------+----------+
  >64  !       WOTJSB        !   TABLE AREA WAIT QUEUE
       +----------+----------+
  >66  !       PDSOLD        !   PRIORITY DSR SCHEDULER QUE HEAD
       +----------+----------+
  >68  !       PDSNEW        !   PRIORITY DSR SCHEDULER QUE TAIL
       +----------+----------+
```

EQUATES:

LABEL	EQUATE TO	VALUE	DESCRIPTION
NFPSIZ	$-TDLHDR	>6A	

+

```
************************************************************
*                                                          *
*      PMDATA - GLOBAL DATA VALUES              04/09/81   *
*               FOR PROGRAM MANAGEMENT                     *
*                                                          *
************************************************************
* THIS COMMON SEGMENT CONTAINS THE PROGRAM MANAGEMENT ERROR
* RECOVERY SAVE AREA (IN THIS AREA SEG1SB THROUGH SEGID2
* MUST BE CONTIGUOUS), THE COMMON DATA FOR TASK BID
* ROUTINES, THE COMMON DATA FOR TASK LOADER ROUTINES, THE
* PROGRAM FILE DIRECTORY DOOR, THE PATHNAME FOR THE SYSTEM
* PROGRAM FILE, AND THE PATHNAME FOR THE SHARED PROGRAM FILE.


     *----------+----------*
 >00 !        SEG1SB       !   SEGMENT 1 SSB ADDRESS
     +----------+----------+
 >02 !        SEG1ST       !   SEGMENT 1 SM TABLE SSB ADDRESS
     +----------+----------+
 >04 !        SEG2SB       !   SEGMENT 2 SSB ADDRESS
     +----------+----------+
 >06 !        SEG2ST       !   SEGMENT 2 SM TABLE SSB ADDRESS
     +----------+----------+
 >08 !        SEG3SB       !   SEGMENT 3 SSB ADDRESS
     +----------+----------+
 >0A !        SEG3ST       !   SEGMENT 3 SM TABLE SSB ADDRESS
     +----------+----------+
 >0C !        CPYLDT       !   ADDRESS OF LDT COPY
     +----------+----------+
 >0E !        CPYRPB       !   ADDRESS OF RPB COPY
     +----------+----------+      ***************************
 >10 !        SEGID1       !   SEGMENT INSTALLED ID(2 WORDS)
     +----------+----------+
 >12 !        SEGID2       !
     +----------+----------+
 >14 !        ATRSG1       !   ATTRIBUTES OF 1ST ATT. SEG.
     +----------+----------+
 >16 !        ATRSG2       !   ATTRIBUTES OF 2ND ATT. SEG.
     +----------+----------+
 >18 !        ATRTSK       !   ATTRIBUTES OF TASK SEG.
     +----------+----------+
 >1A !        LENSG1       !   BYTE LENGTH OF 1ST ATT. SEG.
     +----------+----------+
 >1C !        LENSG2       !   BYTE LENGTH OF 2ND ATT. SEG.
     +----------+----------+
 >1E !        LENTSK       !   BYTE LENGTH OF TASK SEG.
     +----------+----------+
 >20 !        LODTSK       !   LOAD ADDRESS OF TASK SEG.
     +----------+----------+
 >22 !        TSKREP       !   SSB REPLICATED IN MEMORY FLAG
     +----------+----------+
 >24 !        JCASSB       !   SSB FOR JCA OF JOB FOR TASK BID
```

```
     +----------+----------+
 >26 !        JCASMT       !   SMT FOR JCA OF JOB FOR TASK BID
     +----------+----------+      ** TASK LOADER COMMON DATA ***
 >28 !        SG1BET       !   BEET ADDRESS OF SEGMENT 1
     +----------+----------+
 >2A !        SG2BET       !   BEET ADDRESS OF SEGMENT 2
     +----------+----------+
 >2C !        SC3BET       !   BEET ADDRESS OF SEGMENT 3
     +----------+----------+
 >2E !        LODFLG       !   FLAG FOR LOADED/NOT LOADED SEG
     +----------+----------+      ***************************
 >30 !        ROLDIR       !   ROLL DIRECTORY POINTER
     +----------+----------+
 >32 !        ROLPRS       !   PHYSICAL RECORD LENGTH ROLL FILE
     +----------+----------+
 >34 !        SYSFDP       !   SYSTEM PF FDP ADDR.
     +----------+----------+
 >36 !        SYSFMT       !   SYSTEM PF FMT ADDR.
     +----------+----------+
 >38 !        SYSFCB       !   SYSTEM PF FCB ADDR.
     +----------+----------+
 >3A !        ROLFDP       !   ROLL FILE FDP ADDR.
     +----------+----------+
 >3C !        ROLFMT       !   FMT OF ROLL FILE
     +----------+----------+
 >3E !        ROLFCB       !   FCB OF ROLL FILE
     +----------+----------+
 >40 !        APLFDP       !   APPLICATION PF FDP ADDR.
     +----------+----------+
 >42 !        APLFMT       !   APPLICATION PF FMT ADDRESS
     +----------+----------+
 >44 !        APLFCB       !   APPLICATION PF FCB ADDRESS
     +----------+----------+
 >46 !        IMGFDP       !   IMAGES PF FDP ADDR.
     +----------+----------+
 >48 !        IMGFMT       !   IMAGES PF FMT ADDR.
     +----------+----------+
 >4A !        IMGFCB       !   IMAGES PF FCB ADDR.
     +----------+----------+
 >4C !        SHRFDP       !   S$SHARED PF FDP ADDR.
     +----------+----------+
 >4E !        SHRFMT       !   S$SHARED PF FMT ADDR.
     +----------+----------+
 >50 !        SHRFCB       !   S$SHARED PF FCB ADDR.
     +----------+----------+
 >52 !        TIMSPN       !   TIME DELAY SVC FOR TASK LOADER
     +----------+----------+      SPIN ON DISK ERRORS
 >54 !                     !
     +----------+----------+
 >56 !        SYSPN        !   PATHNAME SYSTEM UTILITY PROG FL
     +----------+----------+
 >58 !   SYSPNC   !        !
     +----------+----------+
       /          /          /
       /          /          /
       +----------+----------+
  >60  !        SHRPN        !   PATHNAME FOR S$SHARED PROG FL
       +----------+----------+
  >62  !  WOMPRI  !  FILL00  !   PRIORITY OF TASK BEING LOADED
       +----------+----------+      RESERVED
```

```
  EQUATES:

     LABEL    EQUATE TO    VALUE    DESCRIPTION
     -----    ---------    -----    ------------------------------
     PMDSIZ    $-SEG1SB      >64
```

22.3 STRUCTURES FROM THE ATABLE DIRECTORY

```
************************************************************
*                                                          *
*  ACCOUNTING RECORD CONTENTS (ACC)             09/09/83   *
*                                                          *
*           LOCATION:  SYSTEM TABLE AREA OR DISK           *
************************************************************
* THE ACC DESCRIBES THE FORMAT OF ENTRIES ON THE QUEUE FOR
* PROCESSING BY THE ACCOUNTING FORMATTING TASK (LGACCT).
* WITH THE EXCEPTION OF THE QUEUE LINK, THE ENTRIES ARE
* EXACTLY THE SAME WHEN ON DISK IN THE ACCOUNTING LOG FILE.
* EACH BLOCK TYPE HAS ITS OWN SET OF INFORMATION FOLLOWING
* A STANDARD HEADER.  THE EXCEPTION IS IPL (RECORD TYPE 6),
* WHICH USES ONLY THE HEADER INFORMATION.

                            FIXED PART
      *----------+----------*
  >00 !        ACCLNK       !   QUEUE LINK
      +----------+----------+

                            FIELD DESCRIPTOR VARIANT
      *----------+----------*
  >02 !  ACCTYP  !  ACCLEN  !   RECORD TYPE
      +----------+----------+      LENGTH OF RECORD
  >04 !        ACCYRD       !   YEAR/DAY
      +----------+----------+
  >06 !  ACCHOU  !  ACCMIN  !   HOUR
      +----------+----------+      MINUTE
  >08 !  ACCSEC  !  ACCPRI  !   SECOND
      +----------+----------+      PRIORITY
  >0A !        ACCJID       !   JOB ID
      +----------+----------+

                            TYPE 1 - JOB INITIALIZATION
      *----------+----------*
  >0C !  ACCAID  !          !   ACCOUNT ID
      +----------+----------+
      /          /          /
      /          /          /
      +----------+----------+
  >1C !  ACCUID  !          !   USER ID
      +----------+----------+
      /          /          /
      /          /          /
      +----------+----------+
  >24 !  ACCJNM  !          !   JOB NAME
      +----------+----------+
      /          /          /
```

```
       /          /          /
       +----------+----------+

                              TYPE 2 - TASK TERMINATION
       *----------+----------*
  >0C  !  ACCTID  !  ACCTCD  !   TASK ID
       +----------+----------+      TASK TERM CODE
  >0E  !       ACCCPU        !   TASK CPU TIME (CLOCK TICKS)
       +----------+----------+
  >10  !                     !
       +----------+----------+
  >12  !       ACCSVC        !   NUMBER SVC'S ISSUED
       +----------+----------+
  >14  !                     !
       +----------+----------+
  >16  !       ACCIOB        !   NUMBER I/O BYTES TRANFERED
       +----------+----------+
  >18  !                     !
       +----------+----------+
  >1A  !       ACCMEM        !   MAX MEMORY ALLOCATED(BEETS)
       +----------+----------+
  >1C  !       ACCWAL        !   WALL CLOCK EXECUTION TIME
       +----------+----------+
  >1E  !                     !
       +----------+----------+
  >20  !  ACCIID  !  ACCSTN  !   INSTALLED TASK ID
       +----------+----------+      STATION ID
  >22  !       ACCATR        !   TASK ATTRIBUTES
       +----------+----------+
  >24  !  ACCTNM  !          !   TASK NAME
       +----------+----------+
       /          /          /
       /          /          /
       +----------+----------+

                              TYPE 3 - JOB TERMINATION
       *----------+----------*
  >0C  !       ACCJUD        !   JCA AREA USED
       +----------+----------+
  >0E  !       ACCJSZ        !   JCA TOTAL SIZE
       +----------+----------+
  >10  !       ACCJEX        !   JOB EXECUTION TIME
       +----------+----------+
  >12  !                     !
       +----------+----------+

                              TYPE 4 - DEVICE ENTRY
       *----------+----------*
  >0C  !  ACCTPF  !  ACCDTP  !   DEVICE TYPE FLAGS
       +----------+----------+      DEVICE TYPE
  >0E  !  ACCNAM  !          !   DEVICE NAME
       +----------+----------+
  >10  !          !          !
```

```
     +----------+----------+
 >12 !        ACCNRQ       !   NUMBER I/O REQUESTS
     +----------+----------+
 >14 !                     !
     +----------+----------+
 >16 !        ACCTMU       !   RESERVED-TIME USED(MINUTES)
     +----------+----------+

                         TYPE 5 - USER ENTRY
     *----------+----------*
 >0C !  ACCCHR  !          !   USER DATA
     +----------+----------+
     /          /          /
     /          /          /
     +----------+----------+

                         TYPE 7 - COMM ENTRY
     *----------+----------*
 >0C !  ACCCOM  !          !   COMM DATA
     +----------+----------+
     /          /          /
     /          /          /
     +----------+----------+

                         SINGLE ENTITY VARIANT
     *----------+----------*
 >02 !  ACCTXT  !          !
     +----------+----------+
     /          /          /
     /          /          /
     +----------+----------+
```

```
FLAGS FOR FIELD: ACCYRD     #04 - YEAR/DAY

  ACCYER = (XXXXXXX.........) - YEAR (7 BITS)
  ACCDAY = (.......XXXXXXXXX) - DAY (9 BITS)
```

EQUATES:

LABEL	EQUATE TO	VALUE	DESCRIPTION
ACCVNT	$	>02	
ACTJIT	1	>01	JOB INITIALIZATION
ACTTTM	2	>02	TASK TERMINATION
ACTJTM	3	>03	JOB TERMINATION
ACTDET	4	>04	DEVICE ENTRY
ACTUET	5	>05	USER ENTRY
ACTIPL	6	>06	IPL ENTRY
ACCOHD	$	>0C	END OF OVERHEAD
ACCJIZ	$	>2C	
ACCTTZ	$	>2C	

```
     ACCJTZ    $           >14
     ACCDSZ    $           >18
     ACCUSZ    $           >52
     ACCISZ    $           >0C
     ACCCTZ    $           >52
```

```
***********************************************************
*                                                         *
*  ALIAS DESCRIPTOR RECORD    (ADR)            02/28/79   *
*                                                         *
*           LOCATION: DISK                                *
***********************************************************
* THE ADR IS A VARIANT OF A FILE DESCRIPTOR RECORD (FDR),
* USED TO DESCRIBE AN ALIAS FOR A FILE NAME.  THE FIELDS
* MARKED HERE WITH *** ARE IN THE ADR TEMPLATE TO MAINTAIN
* COMPATABILITY WITH THE FDR TEMPLATE.


      *----------+----------*
  >00 !       ADRHKC        !   HASH KEY COUNT
      +----------+----------+
  >02 !       ADRHKV        !   HASH KEY VALUE
      +----------+----------+
  >04 !  ADRFNM  !          !   FILE NAME
      +----------+----------+
      /          /          /
      /          /          /
      +----------+----------+
  >0C !  ADRPSW  !          !   PASSWORD
      +----------+----------+
  >0E !          !          !
      +----------+----------+
  >10 !       ADRFLG        !   FLAGS(SAME AS FDRFLG FLAGS)
      +----------+----------+
  >12 !       FILL00        !   ***  PHYSICAL RECORD SIZE
      +----------+----------+
  >14 !       FILL01        !   ***  LOGICAL RECORD SIZE
      +----------+----------+
  >16 !       FILL02        !   ***  PRIMARY ALLOCATION SIZE
      +----------+----------+
  >18 !       FILL03        !   ***  PRIMARY ALLOCATION ADDRESS
      +----------+----------+
  >1A !       FILL04        !   ***  SECONDARY ALLOCATION SIZE
      +----------+----------+
  >1C !       FILL05        !   ***  SECONDARY ALLOCATION ADDRESS
      +----------+----------+
  >1E !       ADRRNA        !   RECORD NUMBER OF NEXT ADR
      +----------+----------+
  >20 !       ADRRAF        !   RECORD # OF ACTUAL FDR
      +----------+----------+
*************************************************************
*                                                           *
*  BUFFER ADDRESS PACKET       (BAP)          9/30/81       *
*                                                           *
*           LOCATION: SYSTEM AREA                           *
*************************************************************
*  THE BAP IS THE ADDRESS OF AN I/O BUFFER WHICH IPC APPENDS
*  TO A BUFFERED I/O REQUEST.
                         ** BEGINNING PACKED RECORD BAP


      *----------+----------*
  >00 !       BAPSMT        !   POINTER TO SMT SSB
      +----------+----------+
  >02 !       BAPSSB        !   POINTER TO BUFFER SEG. SSB
      +----------+----------+
  >04 !       BAPOFF        !   OFFSET TO BUFFER WITHIN SEG.
      +----------+----------+
  >06 SIZE               ** END OF PACKED RECORD
```

```
************************************************************
*                                                          *
*  BUFFERED REQUEST OVERHEAD  (BRO)            09/09/83   *
*                                                          *
*           LOCATION: SYSTEM TABLE AREA AND JCA            *
************************************************************
* THE BRO APPEARS AT THE HEAD OF EACH BUFFERED SVC REQUEST
* BLOCK WHILE BEING PROCESSED BY DNOS.  THE REQUEST IS
* QUEUED USING THE BROBRO FIELD.


      *----------+----------*
>FFEE !  BROOFL  !  BROPRI  !   OVERHEAD FLAGS
      +----------+----------+      TASK PRIORITY
>FFF0 !  BROOF2  !  BROAID  !   OVERHEAD FLAGS PART 2
      +----------+----------+      ALTERNATE REQUEST ID FOR M/D DSR
>FFF2 !       BROBBA        !   BUFFER BEET ADDRESS
      +----------+----------+
>FFF4 !       BROLDT        !   LDT ADDRESS
      +----------+----------+
>FFF6 !       BROSID        !   SESSION/DEVICE ID
      +----------+----------+
>FFF8 !       BRORCB        !   REQUESTOR CALL BLOCK ADDRESS
      +----------+----------+
>FFFA !       BROTSB        !   TSB ADDRESS
      +----------+----------+
>FFFC !       BROJSB        !   JSB ADDRESS
      +----------+----------+
>FFFE !       BROBRO        !   QUEUE LINK ADDRESS
      +----------+----------+


  FLAGS FOR FIELD: BROOFL     #FFEE - OVERHEAD FLAGS

    BRFINR = (X...............) - INITIATE EVENT REQUEST
    BRFARS = (.X..............) - ANOTHER ROUTINE HAS SEEN REQ
    BRFA5  = (..X.............) - >A5 CALL GIVEN TO IPC (IURL)
    BRFERN = (...XXXXX........) - INITIATE REQUEST NUMBER


  FLAGS FOR FIELD: BROOF2     #FFF0 - OVERHEAD FLAGS PART 2

    BRFAPI = (X...............) - ALTERNATE REQUEST ID SPECIFIED
    BRFRAV = (.X..............) - REQUEST ACCEPTS EVENT KEYS
    BRFMRO = (..X.............) - MULTI-RECORD READ/WRITE REQUEST
    BRFTID = (...X............) - TASK ID SPECIFIED IN BROTSB
    BRFSB  = (....X...........) - SECURITY BYPASS
    BRFSAB = (.....X..........) - SUSPENDING ABORT
    BRFDNR = (......X.........) - DO NOT RELEASE MEMORY
           = (.......X........) - UNUSED
```

EQUATES:

LABEL	EQUATE TO	VALUE	DESCRIPTION
BRFEOR	BRFA5	>02	END OF RECORD DONE IMMEDIATELY
BRFABT	BRFSAB	>05	ABORTED OPERATION
BROSIZ	$-BROOFL	>12	

```
************************************************************
*                                                          *
*  B - TREE BLOCK             (BTB)             09/07/79  *
*                                                          *
*           LOCATION: DISK AND BUFFER SEGMENTS             *
************************************************************
* THE BTB DESCRIBES THE OVERHEAD INFORMATION REQUIRED TO SORT
* THE LOGICAL RECORDS OF A KEY INDEXED FILE.  THE BTB RESIDES
* ON DISK AND IS READ INTO MEMORY WHEN USING A RECORD THAT
* IT DESCRIBES.
*
* SPECIAL FIELD COMMENTS:
* BTBPPT - IF THIS BLOCK IS BEING USED AS A B-TREE NODE, THIS
*          FIELD IS THE PHYSICAL RECORD NUMBER OF THE
*          PRECEDING NODE ON THE SAME LEVEL (ZERO IF THIS IS
*          THE LEFTMOST NODE).  IF THIS BLOCK IS AVAILABLE
*          FOR USE, THIS FIELD POINTS TO THE NEXT AVAILABLE
*          BLOCK.
* BTBNIE - NUMBER OF POINTER/KEY VALUE PAIRS CURRENTLY
*          CONTAINED IN THIS BLOCK.
* BTBNEA - THIS BYTE IS ZERO WHEN THE BLOCK IS INITIALIZED
*          BECAUSE OF A B-TREE SPLIT.  WHEN THE FIRST ENTRY
*          IS MADE TO THE BLOCK, THIS BYTE CONTAINS THE
*          NUMBER OF ENTRIES IN THE BLOCK THAT ARE GREATER
*          THAN THE NEW ENTRY.
* BTBNEC - WHEN THE BLOCK IS INITIALIZED DUE TO A B-TREE
*          SPLIT, THIS VALUE IS THE MAXIMUM ENTRIES THAT MAY
*          BE INSERTED INTO THE BLOCK, PLUS ONE.  FOR EACH
*          SUBSEQUENT ENTRY TO THIS BLOCK, IF THE NUMBER OF
*          ENTRIES IN THE BLOCK THAT ARE GREATER THAN THE NEW
*          ENTRY EQUALS THE NUMBER IN BTBNEA, BTBNEC IS
*          DECREMENTED BY ONE.  WHEN THIS B-TREE BLOCK
*          IS ABOUT TO SPLIT, IF BTBNEC IS ZERO, THE SPLIT IS
*          AT A RATIO OF THE LOWER 90% OF THE ENTRIES ARE IN
*          ONE BLOCK AND THE UPPER 10% IN THE OTHER.  OTHERWISE,
*          THE SPLIT IS 50% TO EACH.
* BTBDBK - IF THIS IS A NON-LEAF NODE, THE FIRST FOUR BYTES
*          CARRY THE RECORD NUMBER OF A BRANCH OR LEAF NODE AND
*          THE LAST TWO BYTES ARE NOT MEANINGFUL.  IF THIS IS A
*          LEAF NODE, THE FIRST FOUR BYTES CONTAIN A RECORD NUMBER
*          OF A DATA RECORD AND THE LAST TWO BYTES CONTAIN THE
*          ID OF THE LOGICAL RECORD WITHIN THE DATA RECORD.
* BTBCMD - THIS FIELD IS USED WHEN THIS RECORD HAS TO BE PRELOGGED.
*          IT IDENTIFIES ALL THE RECORDS PRELOGGED BY THE OPERATION.


     *----------+----------*
 >00 !       BTBBLK        !   BLOCK NUMBER (2 WORD PHYSICAL
     +----------+----------+      RECORD NUMBER OF THIS BTB)
 >02 !                     !
     +----------+----------+
 >04 !       BTBCMD        !   PRELOG NUMBER
```

```
     +----------+----------+
>06  !        BTBSR        !   SPACE REMAINING (BYTES) IN BTB
     +----------+----------+
>08  !        BTBPPT       !   PREDECESSOR OR FREE POINTER
     +----------+----------+
>0A  !                     !
     +----------+----------+
>0C  !        BTBSPT       !   SUCCESSOR POINTER
     +----------+----------+
>0E  !                     !
     +----------+----------+
>10  !  BTBNIE  !  BTBLE   !   NUMBER OF INDEX ENTRIES
     +----------+----------+      LEAF ENTRY FLAG(1=THIS IS A LEAF)
>12  !  BTBNEA  !  BTBNEC  !   # OF ENT. AFTER LAST INSERT
     +----------+----------+      COUNT OF # SEQ. INSERTS
>14  !  BTBDBK  !          !   B-BLOCK DATA BASE KEY
     +----------+----------+
>16  !          !          !
     +----------+----------+
>18  !          !          !
     +----------+----------+
>1A  !        BTBKVL       !   FIRST POINTER/KEY VALUE PAIR
     +----------+----------+
```

EQUATES:

```
   LABEL    EQUATE TO    VALUE    DESCRIPTION
   -----    ---------    -----    -----------------------------
   BTBSIZ   $              >1C
```

```
***********************************************************
*                                                         *
*  CHANNEL CONTROL BLOCK        (CCB)          06/09/83  *
*                                                         *
*           LOCATION: SYSTEM AREA AND JCA                 *
***********************************************************
* THE CCB IS THE IN-MEMORY REPRESENTATION OF A CHANNEL.  IT
* RESIDES IN SYSTEM TABLE AREA FOR GLOBAL CHANNELS, IN THE
* JCA FOR JOB-LOCAL OR TASK-LOCAL CHANNELS.  MOST OF THIS
* STRUCTURE IS BUILT FROM THE CHANNEL DESCRIPTOR RECORD ON
* DISK.


      *----------+----------*
  >00 !        CCBCCB       !    NEXT CCB ADDRESS
      +----------+----------+
  >02 !        CCBFLG       !    CHANNEL FLAGS
      +----------+----------+
  >04 !  CCBTYP  !  CCBTF   !    DEFAULT RESOURCE TYPE
      +----------+----------+       RESOURCE TYPE FLAGS
  >06 !        CCBMXL       !    MAXIMUM MESSAGE LENGTH
      +----------+----------+
  >08 !  CCBASG  !  CCBOPN  !    NUMBER OF CURRENT ASSIGNS
      +----------+----------+       NUMBER OF CURRENT OPENS
  >0A !        CCBRPB       !    RPB POINTER
      +----------+----------+
  >0C !        CCBTSB       !    OWNER TASK TSB ADDRESS
      +----------+----------+
  >0E !        CCBJSB       !    OWNER TASK JSB ADDRESS
      +----------+----------+
  >10 !  CCBPFL  !  CCBIID  !    OWNER TASK PROG FILE LUNO (DPOS/M)
      +----------+----------+       OWNER TASK INSTALLED ID
  >12 !        CCBFMT       !    OWNER TASK PROGRAM FILE FDP (D)
      +----------+----------+
  >14 !        CCBFCB       !
      +----------+----------+
  >16 !        CCBPBQ       !    PENDING BRB QUEUE HEADER
      +----------+----------+
  >18 !        CCBABQ       !    ALREADY BEING PROCESSED QUEUE HEAD
      +----------+----------+
  >1A !  CCBNAM  !          !    CHANNEL NAME LENGTH AND NAME (M).
      +----------+----------+
      /          /          /
      /          /          /
      +----------+----------+


  FLAGS FOR FIELD: CCBFLG     #02 - CHANNEL FLAGS

    CCFSC1 = (XX..............) - SCOPE - GLOBAL,JOB,TASK
*                                 00=TASK-LOCAL
*                                 01=JOB-LOCAL
*                                     10=GLOBAL
*                                     11=RESERVED
    CCFSHR = (..X.............) - SHARED(1)/NOT SHARED
    CCFTYP = (...X............) - SYMMETRIC(1) OR MASTER/SLAVE
    CCFASG = (....X...........) - OWNER DOES(1) / NOT DO ASSIGN
    CCFABT = (.....X..........) - OWNER DOES(1) / NOT DO ABORTS
    CCFIOU = (......X.........) - OWNER DOES(1) / NOT DO IOU OPS
           = (.......X........) - RESERVED (AS IN CREATE CHAN RCB)
    CCFBSY = (........X.......) - CCB IS BUSY (IN USE BY IPC TASK)
    CCFOOP = (.........X......) - OWNER TASK HAS ISSUED OPEN
    CCFOCL = (..........X.....) - OWNER TASK HAS CLOSED OR ABORTED
    CCFDED = (...........X....) - CHANNEL IS DEAD
    CCFRCL = (............X...) - NON-SH SYMMETRIC REQUESTER CLOSED
    CCFRAB = (.............X..) - NON-SH SYMMETRIC REQUESTER ABORTED
           = (..............XX) - RESERVED
*
*                              NOTE: CCBTF=CCBTYP+1 MUST BE TRUE
```

EQUATES:

LABEL	EQUATE TO	VALUE	DESCRIPTION
CCFSC2	CCFSC1+1	>01	
CCFSCM	>C000	>C000	CHANNEL SCOPE MASK
CCFCHN	13	>0D	CHANNEL
CCFDEV	14	>0E	DEVICE
CCFFIL	15	>0F	FILE
CCFREM	>0007	>07	RESOURCE TYPE FLAGS MASK
CCBDSZ	$+8	>22	DPOS/D CCB SIZE
CCBCSZ	$	>4C	DPOS/M CCB SIZE

```
*************************************************************
*                                                           *
*  COMMAND DEFINITION ENTRY   (CDE)             04/02/82    *
*                                                           *
*************************************************************
* THE CDE DESCRIBES ONE ENTRY IN THE COMMAND DEFINITION
* TABLE FOR A DEVICE.  THE ENTRY SHOWS WHAT TASK IS TO BE
* BID WHEN A KEYBOARD TASK BID IS DONE.


     *----------+----------*
 >00 !  CDECHR  !  CDEFLG  !   ENTRY IDENTIFICATION CHARACTER
     +----------+----------+      BID FLAGS
 >02 !  CDELL   !  CDELID  !   LUNO WITH WHICH TO BID LOGIN
     +----------+----------+      LOGIN TASK ID
 >04 !  CDEDL   !  CDEDID  !   LUNO TO BID DESTINATION TASK
     +----------+----------+      DESTINATION TASK ID
 >06 !       CDEPV1        !   PARAMETER VALUE 1 FOR DEST. TASK
     +----------+----------+
 >08 !       CDEPV2        !   PARAMETER VALUE 2 FOR DEST. TASK
     +----------+----------+
 >0A !  CDEUID  !          !   DEFAULT USER ID
     +----------+----------+
     /          /          /
     /          /          /
     +----------+----------+


 FLAGS FOR FIELD: CDEFLG    #01 - BID FLAGS

   CDFBCJ = (X...............) - BID DEST. TASK IN CURRENT JOB
   CDFPEA = (.X..............) - PASS THE CDE ADDRESS TO LOGIN
   CDFELB = (..X.............) - EVEN LOADING BID
   CDFDUI = (...X............) - DEFAULT USER ID FOR ELB
          = (....XXXX........) - * RESERVED *


 EQUATES:

    LABEL   EQUATE TO   VALUE   DESCRIPTION
    -----   ---------   -----   ------------------------------
    CDESIZ   $            >12    COMMAND DEFINITION ENTRY SIZE
```

```
***********************************************************
*                                                         *
*  CHANNEL DESCRIPTOR RECORD   (CDR)            08/14/81  *
*                                                         *
*           LOCATION: DISK                                *
***********************************************************
* THE CDR IS THE PERMANENT RECORD OF A CHANNEL.  IT IS
* CARRIED AS AN ALIAS OF THE PROGRAM FILE IN WHICH THE
* CHANNEL OWNER TASK RESIDES.


      *----------+----------*
  >00 !        CDRHKC       !   HASH KEY COUNT
      +----------+----------+
  >02 !        CDRHKV       !   HASH KEY VALUE
      +----------+----------+
  >04 !  CDRNAM  !          !   CHANNEL NAME
      +----------+----------+
      /          /          /
      /          /          /
      +----------+----------+
  >0C !        FILL00       !   RESERVED
      +----------+----------+
  >0E !        FILL01       !   RESERVED
      +----------+----------+
  >10 !        CDRFDF       !   FLAGS
      +----------+----------+
  >12 !  CDRFLG  !  CDRIID  !   CHANNEL FLAGS
      +----------+----------+      OWNER TASK INSTALLED ID
  >14 !  CDRTYP  !  CDRTF   !   DEFAULT RESOURCE TYPE
      +----------+----------+      RESOURCE TYPE FLAGS
  >16 !        CDRMXL       !   MAXIMUM MESSAGE LENGTH
      +----------+----------+
  >18 !  FILL04  !          !   RESERVED
      +----------+----------+
  >1A !          !          !
      +----------+----------+
  >1C !          !          !
      +----------+----------+
  >1E !        CDRRNA       !   RECORD NUMBER OF NEXT CDR OR ADR
      +----------+----------+
  >20 !        CDRRAF       !   RECORD NUMBER OF ACTUAL FDR
      +----------+----------+
  >22 !  FILL05  !          !   RESERVED
      +----------+----------+
      /          /          /
      /          /          /
      +----------+----------+
  >90 !  CDRUID  !          !   USER ID OF CHANNEL CREATOR
      +----------+----------+
      /          /          /
      /          /          /
      +----------+----------+
 >98 !      CDRPSA         !    PUBLIC SECURITY ATTRIBUTES
      +----------+----------+
 >9A !  CDRSCG  !          !    SDT WITH 9 CONTROL GROUPS
      +----------+----------+
      /          /          /
      /          /          /
      +----------+----------+
 >F8 !  FILL06  !          !    RESERVED
      +----------+----------+
      /          /          /
      /          /          /
      +----------+----------+


 FLAGS FOR FIELD: CDRFDF     #10 - FLAGS

          = (XXXXXXXXXXXXXXX.) - STANDARD FDR FLAGS
   CDFCDR = (...............X) - CDR(1) OR NOT(0)


 FLAGS FOR FIELD: CDRFLG     #12 - CHANNEL FLAGS

   CDFSC1 = (XX..............) - SCOPE - GLOBAL, JOB, TASK
*                              00=TASK-LOCAL
*                              01=JOB-LOCAL
*                              10=GLOBAL
*                              11=RESERVED
   CDFSHR = (..X.............) - SHARED(1) OR NOT SHARED
   CDFTYP = (...X............) - SYMMETRIC(1) OR MASTER/SLAVE
   CDFASG = (....X...........) - OWNER DOES(1) / NOT DO ASSIGN
   CDFABT = (.....X..........) - OWNER DOES(1) / NOT DO ABORTS
   CDFIOU = (......X.........) - OWNER DOES(1) / NOT DO IOU OPS
          = (.......X........) - RESERVED (AS CREATE CHANNEL)


 EQUATES:

    LABEL   EQUATE TO   VALUE   DESCRIPTION
    -----   ---------   -----   ------------------------------
    CDRDPM   >0080        >80    DELETE-PROTECT MASK
    CDRCDM   >0001        >01    CDR FLAG MASK
    CDFSC2   CDFSC1+1     >01
    CDFSCM   >C000        >C000    MASK FOR CHANNEL SCOPE
    CDFRM1   >FE00        >FE00    MASK TO ZERO RESERVED BITS
    CDFCHN   13           >0D    CHANNEL
    CDFDEV   14           >0E    DEVICE
    CDFFIL   15           >0F    FILE
    CDFRM2   >FF07        >FF07    MASK TO ZERO RESERVED TYPE FLAGS
    CDRSIZ   $            >100
    CDRMAX   >3000        >3000    MAXIMUM VALUE FOR CDRMXL
```

```
***********************************************************
*                                                         *
*      CAPABILITIES LIST FILE RECORD (CLR)      01/21/83  *
*                                                         *
*         LOCATION .S$CLF ON DISK                         *
*                                                         *
***********************************************************
* THE CLR IS USED BY TASKS WHICH ADD, DELETE, OR MODIFY
* USER IDS OR ACCESS GROUPS.  IT HAS 5 VARIANTS: FIR, AGR,
* UDR, UDO, AND VFY.  THE STRUCTURE AND PURPOSE OF EACH VARIANT
* IS DESCRIBED BELOW.
*
*
*  THIS PACKED RECORD IS USED FOR USER ID ENTRIES IN FIR
*
                         ** BEGINNING PACKED RECORD UID


      *----------+----------*
  >00 !  FIRID   !          !   USER ID
      +----------+----------+
      /          /          /
      /          /          /
      +----------+----------+
  >08 !       FIRRN         !   USER'S UDR RECORD NUMBER
      +----------+----------+
  >0A SIZE               ** END OF PACKED RECORD

*
*  THIS PACKED RECORD IS USED FOR ACCESS GROUP ENTRIES IN
*  USER DESCRIPTOR RECORDS (UDR) AND USER DESCRIPTOR OVERFLOW
*  RECORDS (UDO).
*
                         ** BEGINNING PACKED RECORD AGE

                            ACCESS GROUP ENTRY
      *----------+----------*
  >00 !       AGERN         !   ACCESS GROUP RECORD NUMBER
      +----------+----------+
  >02 !  AGEOFF  !  AGEFLG  !   OFFSET INTO ACCESS GROUP RECORD
      +----------+----------+      ACCESS GROUP ENTRY FLAGS
  >04 SIZE               ** END OF PACKED RECORD

*
*  THIS PACKED RECORD IS USED FOR ACCESS GROUP NAMES IN
*  ACCESS GROUP RECORDS (AGR)
*
                         ** BEGINNING PACKED RECORD AGN


      *----------+----------*
  >00 !  AGNNAM  !          !   ACCESS GROUP NAME
```

```
          +----------+----------+
          /          /          /
          /          /          /
          +----------+----------+
     >08  !       AGNRSV        !   RESERVED
          +----------+----------+
     >0A  SIZE                ** END OF PACKED RECORD

*
                          ** BEGINNING PACKED RECORD CLR
*
*
*
*           FILE INFORMATION RECORD (FIR)
*
*   THIS VARIANT IS USED TO STORE USER IDs.  IT CONTAINS A
*   FLAG WORD, A POINTER TO ANOTHER FIR, AND 5 UID ENTRIES.
*   EACH UID ENTRY CONTAINS A USER ID AND THE RECORD NUMBER
*   OF ITS USER DESCRIPTOR RECORD (UDR).
*
*


          *----------+----------*
     >00  !       FIRFIR        !   CONTINUATION RECORD NUMBER
          +----------+----------+
     >02  !       FIRRSV        !   FIR USED/AVAILABLE FLAG
          +----------+----------+
     >04  !       FIRENT        !   5 UID ENTRIES
          +----------+----------+
          /          /          /
          /          /          /
          +----------+----------+
     >0E  !                     !
          +----------+----------+
          /          /          /
          /          /          /
          +----------+----------+
     >18  !                     !
          +----------+----------+
          /          /          /
          /          /          /
          +----------+----------+
     >22  !                     !
          +----------+----------+
          /          /          /
          /          /          /
          +----------+----------+
     >2C  !                     !
          +----------+----------+
          /          /          /
          /          /          /
          +----------+----------+
*
*         ACCESS GROUP NAME RECORD (AGR)
*
*  THIS VARIANT IS USED TO STORE ACCESS GROUP NAMES.
*  IT CONTAINS A FLAG WORD, A POINTER TO THE NEXT AGR, AND
*  5 AGN ENTRIES.  EACH AGN ENTRY CONTAINS AN ACCESS GROUP
*  NAME AND A WORD OF UNUSED FLAGS.
*


      *----------+----------*
  >00 !        AGRAGR       !   CONTINUATION RECORD NUMBER
      +----------+----------+
  >02 !        AGRRSV       !   AGR USED/AVAILABLE FLAG
      +----------+----------+
  >04 !        AGRAGN       !   5 AGN ENTRIES
      +----------+----------+
      /          /          /
      /          /          /
      +----------+----------+
  >0E !                     !
      +----------+----------+
      /          /          /
      /          /          /
      +----------+----------+
  >18 !                     !
      +----------+----------+
      /          /          /
      /          /          /
      +----------+----------+
  >22 !                     !
      +----------+----------+
      /          /          /
      /          /          /
      +----------+----------+
  >2C !                     !
      +----------+----------+
      /          /          /
      /          /          /
      +----------+----------+
*
*         USER DESCRIPTOR RECORD (UDR)
*
*  THIS VARIANT CONTAINS INFORMATION ASSOCIATED WITH A USER ID.
*  THIS INFORMATION INCLUDES THE ENCRYPTED PASSCODE, DESCRIPTION,
*  AND UP TO 5 ACCESS GROUP ENTRIES.  EACH ACCESS GROUP ENTRY
*  CONTAINS A RECORD NUMBER OF AN ACCESS GROUP RECORD (AGR)
*  AND THE OFFSET INTO THE AGR FOR AN ACCESS GROUP NAME OF
*  WHICH THIS USER IS A MEMBER.
*


      *----------+----------*
```

```
   >00 !         UDRUDO         !    POINTER TO OVERFLOW
       +-----------+-----------+
   >02 !         UDRRSV         !    UDR USED/AVAILABLE FLAG
       +-----------+-----------+
   >04 !  UDRPWD   !           !    ENCRYPTED PASSCODE
       +-----------+-----------+
       /           /           /
       /           /           /
       +-----------+-----------+
   >0C !         UDRFLG         !    UDR FLAG WORD
       +-----------+-----------+
   >0E !  UDRDES   !           !    DESCRIPTION OF USER
       +-----------+-----------+
       /           /           /
       /           /           /
       +-----------+-----------+
   >22 !         UDRAGE         !    5 ACCESS GROUP ENTRIES (AGE)
       +-----------+-----------+
   >24 !           !           !
       +-----------+-----------+
   >26 !                       !
       +-----------+-----------+
   >28 !           !           !
       +-----------+-----------+
   >2A !                       !
       +-----------+-----------+
   >2C !           !           !
       +-----------+-----------+
   >2E !                       !
       +-----------+-----------+
   >30 !           !           !
       +-----------+-----------+
   >32 !                       !
       +-----------+-----------+
   >34 !           !           !
       +-----------+-----------+
*
*        USER DESCRIPTOR OVERFLOW RECORD (UDO)
*
*  THIS VARIANT IS USED ONLY USED IN THE CASE THAT A USER IS
*  A MEMBER OF MORE ACCESS GROUPS THAN WILL FIT IN HIS UDR.
*  IT CONTAINS UP TO 12 ACCESS GROUP ENTRIES.
*


       *-----------+-----------*
   >00 !         UDOUDO         !    POINTER TO NEXT UDO
       +-----------+-----------+
   >02 !         UDORSV         !    UDO USED/AVAILABLE FLAG
       +-----------+-----------+
   >04 !         UDOFIL         !    NOT USED
       +-----------+-----------+
   >06 !         UDOAGE         !    12 ACCESS GROUP ENTRIES (AGE)
```

```
      +----------+----------+
  >08 !          !          !
      +----------+----------+
  >0A !                     !
      +----------+----------+
  >0C !          !          !
      +----------+----------+
  >0E !                     !
      +----------+----------+
  >10 !          !          !
      +----------+----------+
  >12 !                     !
      +----------+----------+
  >14 !          !          !
      +----------+----------+
  >16 !                     !
      +----------+----------+
  >18 !          !          !
      +----------+----------+
  >1A !                     !
      +----------+----------+
  >1C !          !          !
      +----------+----------+
  >1E !                     !
      +----------+----------+
  >20 !          !          !
      +----------+----------+
  >22 !                     !
      +----------+----------+
  >24 !          !          !
      +----------+----------+
  >26 !                     !
      +----------+----------+
  >28 !          !          !
      +----------+----------+
  >2A !                     !
      +----------+----------+
  >2C !          !          !
      +----------+----------+
  >2E !                     !
      +----------+----------+
  >30 !          !          !
      +----------+----------+
  >32 !                     !
      +----------+----------+
  >34 !          !          !
      +----------+----------+
*
*        VERIFICATION RECORD (VFY)
*
*  THIS VARIANT IS USED BY THE SYSTEM RESTART TASK TO VERIFY
*  THE EXISTENCE OF .S$CLF.  IT IS ALSO USED BY TASKS WHICH
*  CREATE AND MODIFY ACCESS GROUPS BECAUSE IT CONTAINS A
*  POINTER TO THE FIRST ACCESS GROUP RECORD.
*


     *----------+----------*
>00  !  VFYNAM  !          !   NAME OF S$CLF
     +----------+----------+
     /          /          /
     /          /          /
     +----------+----------+
>08  !       VFYBLK        !   POINTER TO FIRST AGRBLK
     +----------+----------+
>0A  !  VFYFIL  !          !   NOT USED, INITIALIZED TO BLANKS
     +----------+----------+
     /          /          /
     /          /          /
     +----------+----------+
>36  SIZE                ** END OF PACKED RECORD


  FLAGS FOR FIELD: AGEFLG     #03 - ACCESS GROUP ENTRY FLAGS

    AGELDR = (X...............) - TRUE=USER IS LEADER OF ACCESS GROUP
    AGEFCG = (.X..............) - TRUE=FILE CREATION ACCESS GROUP


  FLAGS FOR FIELD: FIRRSV     #02 - FIR USED/AVAILABLE FLAG

    FIRFRE = (X...............) - TRUE=AVAILABLE RECORD


  FLAGS FOR FIELD: AGRRSV     #02 - AGR USED/AVAILABLE FLAG

    AGRFRE = (X...............) - TRUE=AVAILABLE RECORD


  FLAGS FOR FIELD: UDRRSV     #02 - UDR USED/AVAILABLE FLAG

    UDRFRE = (X...............) - TRUE=AVAILABLE RECORD


  FLAGS FOR FIELD: UDRFLG     #0C - UDR FLAG WORD

    UDRPVL = (XXXXX...........) - USER PRIVELEDGE LEVEL
    UDRAGC = (.....XXXXXXXXXXX) - ACCESS GROUP COUNT


  FLAGS FOR FIELD: UDORSV     #02 - UDO USED/AVAILABLE FLAG

    UDOFRE = (X...............) - TRUE=AVAILABLE ENTRY
```

EQUATES:

LABEL	EQUATE TO	VALUE	DESCRIPTION
FIR	$	>00	
AGR	$	>00	
UDR	$	>00	
UDO	$	>00	
VFY	$	>00	
FIRSIZ	$	>36	
AGRSIZ	$	>36	
UDRSIZ	$	>36	
UDOSIZ	$	>36	
VFYSIZ	$	>36	

```
*************************************************************
*                                                           *
*  DIOU DATA BASE DEFINITION   (DDB)              12/01/81*
*                                                           *
*           LOCATION: DIOU NAME MANAGER SEGMENTS AND        *
*                        RELATIVE RECORD FILE               *
*************************************************************
                         ** BEGINNING PACKED RECORD DDB
*                         DEVICE NAMES
*                         DEVICE NUMBERS
*
*       RELATIVE RECORD FILE RECORDS
*


      *----------+----------*
  >00 !       DDB0R1        !   *RESERVED
      +----------+----------+
  >02 !       DDB0RT        !   *RESOURCE TYPE
      +----------+----------+
  >04 !  DDB0DT  !  DDB0CT  !   *DEVICE TYPE
      +----------+----------+      *CDT NUMBER
  >06 !       DDB0CE        !   *CDE MASK
      +----------+----------+
  >08 !  DDB0WT  !  DDB0R2  !   *WRITE TASK ID
      +----------+----------+      *RESERVED


      *----------+----------*
  >00 !       DDB1NU        !   *DEVICE NUMBER
      +----------+----------+
  >02 !       DDB1RR        !   *RELATIVE RECORD NUMBER
      +----------+----------+
  >04 !       DDB1PA        !   *PDT ADDRESS
      +----------+----------+
  >06 !       DDB1F1        !   *FLAGS
      +----------+----------+
  >08 !       DDB1F2        !   *FLAGS
      +----------+----------+
  >0A !  DDB1LC  !  DDB1TC  !   *ASSIGNED LUNO COUNT
      +----------+----------+      *ATTACHED TASK COUNT
  >0C !       DDB1OJ        !   *OWNER JOB
      +----------+----------+
  >0E !       DDB1LP        !   *LOCKED PARAMETER LIST ANCHOR
      +----------+----------+
  >10 !       DDB1RP        !   *RPB ANCHOR
      +----------+----------+


      *----------+----------*
  >00 !  DDB2R1  !  DDB2NF  !   *RESERVED
      +----------+----------+      *DEVICE NAME FILE
```

```
>02 !        DDB2RR         !   *RELATIVE RECORD NUMBER
    +----------+----------+
>04 !  DDB2NA  !                  *DEVICE NAME
    +----------+


    *----------+----------*
>00 !  OSPPRM  !  OSPSTR  !   *PARAMETER NUMBER
    +----------+----------+      *STRUCTURE IDENTIFIER
>02 !  OSPOFF  !  OSPLEN  !   *OFFSET INTO STRUCTURE
    +----------+----------+      *LENGTH OF PARAMETER


    *----------+----------*
>00 !  STRES   !  STID    !   *RESERVED
    +----------+----------+      *SESSION IDENTIFIER
>02 !        STFNAS         !   *FIRST NAME SEGMENT RUN ID
    +----------+----------+
>04 !        STFNUS         !   *FIRST NUMBER SEGMENT RUN ID
    +----------+----------+
>06 !  FILL00  !          !   *REST OF THE RUN IDS
    +----------+----------+
    /          /          /
    /          /          /
    +----------+----------+
>1A !  STPTNM  !          !   *PATHNAME OF SYSTEM
    +----------+----------+
    /          /          /
    /          /          /
    +----------+----------+
>2C SIZE                ** END OF PACKED RECORD
```

EQUATES:

```
   LABEL    EQUATE TO    VALUE    DESCRIPTION
   -----    ---------    -----    ------------------------------
   RELREC   0               >00     *RELATIVE RECORD FILE
   NAMSEG   1               >01     *NAME MANAGER SEGMENTS ORDERED BY
   NUMSEG   2               >02     *NAME MANAGER SEGMENTS ORDERED BY
   DSKPDT   3               >03     *DISK PDT
   DDBOVL   $               >0A     *BEGINNING OF VARIABLE LENGTH PARMS
   DDBOPD   $               >0A     *PRINT DEVICE NAME
   DDBOVD   $               >0A     *VIRTUAL DEVICE SERVER
   DDBOUP   $               >0A     *USER PARAMETERS
   DDBNOS   $               >0A     *NON O.S. PARAMETERS
   DDB1SZ   $               >12     *SIZE
   DDF1DS   >1800           >1800     *DEVICE STATE MASK
```

```
***********************************************************
*                                                         *
*  DIAGNOSTIC STATUS           (DIA)            05/16/79  *
*                                                         *
*            LOCATION:  JCA                               *
***********************************************************
* THE DIA DESCRIBES A TASK WHICH IS TERMINATING ABNORMALLY.
* IT IS USED TO PROVIDE END ACTION STATUS TO A TASK AND TO
* BUILD A TERMINATION MESSAGE FOR THE SYSTEM LOG.


      *----------+----------*
  >00 !  DIAEC    !  FILL00  !   TASK ERROR CODE
      +----------+----------+       RESERVED
  >02 !        DIAWP        !   TASK WORKSPACE POINTER
      +----------+----------+
  >04 !        DIAPC        !   TASK PROGRAM COUNTER
      +----------+----------+
  >06 !        DIAST        !   TASK STATUS
      +----------+----------+
  >08 !        DIALM1       !   END ACTION TIME LIMIT(1ST WORD)
      +----------+----------+
  >0A !        DIALM2       !   (SECOND WORD)
      +----------+----------+

  EQUATES:

     LABEL   EQUATE TO   VALUE   DESCRIPTION
     -----   ---------   -----   ------------------------------
     DIASIZ  $              >0C
```

```
********************************************************************
*                                                                  *
*           DISK INFORMATION TABLE    (DIT)               09/09/83 *
*                                                                  *
********************************************************************
* THIS TEMPLATE IS USED TO DESCRIBE EACH ENTRY IN THE DITDAT TABLE
* USED BY THE DISK VOLUME UTILITIES.  THERE IS ONE ENTRY IN DITDAT
* FOR EACH DISK TYPE THAT IS SUPPORTED BY DNOS.
*

                              ** BEGINNING PACKED RECORD DIT


      *----------+----------*
  >00 !        DITSR1       !   DISK STORE REGISTERS 1
      +----------+----------+
  >02 !        DITSR2       !   DISK STORE REGISTERS 2
      +----------+----------+
  >04 !        DITSR3       !   DISK STORE REGISTERS 3
      +----------+----------+
  >06 !        DITFLG       !   DISK INFORMATION FLAGS
      +----------+----------+
  >08 !        DITPRL       !   PHYSICAL RECORD LENGTH-DEFAULT
      +----------+----------+
  >0A !        DITNVE       !   NUMBER VCATALOG ENTRIES-DEFAULT
      +----------+----------+
  >0C !  DITNAM  !          !   DISK NAME
      +----------+----------+
      /          /          /
      /          /          /
      +----------+----------+
  >1C !  DITNSC  !  DITSCM  !   NUMBER SPARE CYLINDERS
      +----------+----------+      SPARE CYLINDERS FOR MAPPING
  >1E !        DITHIF       !   HARDWARE INTERLEAVE FACTOR-DEFAULT
      +----------+----------+
  >20 !        DITTPP       !   TEST PATTERNS POINTER
      +----------+----------+
  >22 !        DITDSP       !   DIAGNOSTIC SECTORS POINTER
      +----------+----------+
  >24 !  DITNDS  !  FILL03  !   NUMBER OF DIAGNOSTIC SECTORS
      +----------+----------+      SPARE BYTE - NOT USED
  >26 !        DITRTF       !   READ TYPES FLAGS FOR SURFACE ANALYSI
      +----------+----------+
  >28 !        DITCRL       !   DISK CONTROLLER REVISION LEVEL
      +----------+----------+
  >2A !        FILL05       !   SPARE - NOT USED
      +----------+----------+
  >2C SIZE                ** END OF PACKED RECORD
```

```
************************************************************
*                                                          *
*  DIRECTORY OVERHEAD RECORD  (DOR)              01/31/79  *
*                                                          *
*            LOCATION:         DISK                        *
************************************************************
* THE DOR IS THE FIRST RECORD (RECORD 0) OF A DIRECTORY FILE
* AND SHOWS THE MAXIMUM SIZE AND CURRENT USE OF A DIRECTORY.
```

```
      *----------+----------*
  >00 !        DORNRC       !   # RECORDS IN DIRECTORY
      +----------+----------+
  >02 !        DORNFL       !   # FILES CURRENTLY IN DIRECTORY
      +----------+----------+
  >04 !        DORNAR       !   # OF AVAILABLE RECORDS
      +----------+----------+
  >06 !        DORTFC       !   NUMBER OF TEMPORARY FILES
      +----------+----------+
  >08 !  DORDNM  !          !   DIRECTORY FILE NAME
      +----------+----------+
      /          /          /
      /          /          /
      +----------+----------+
  >10 !        DORLVL       !   LEVEL # OF DIRECTORY
      +----------+----------+
  >12 !  DORPNM  !          !   NAME OF PARENT FILE
      +----------+----------+
      /          /          /
      /          /          /
      +----------+----------+
  >1A !        DORPRS       !   DEFAULT PHYSICAL RECORD LENGTH
      +----------+----------+      (USED FOR FILE CREATION)
```

EQUATES:

LABEL	EQUATE TO	VALUE	DESCRIPTION
DORSIZ	$	>1C	

```
*************************************************************
*                                                           *
*  DISK PDT EXTENSION         (DPD)               01/17/83  *
*                                                           *
*          LOCATION: SYSTEM TABLE AREA                      *
*************************************************************
* THE DPD APPEARS AFTER THE STANDARD PDT INFORMATION FOR A
* DISK DEVICE. IT IS USED AS A WORK AREA BY THE DSR AND BY
* THE DISK MANAGER TASK.


      *----------+----------*
  >00 !  DPDTIL  !          !   TILINE IMAGE
      +----------+----------+
      /          /          /
      /          /          /
      +----------+----------+
  >10 !  DPDSLG  !          !   TILINE IMAGE FOR SYSTEM LOG
      +----------+----------+
      /          /          /
      /          /          /
      +----------+----------+
  >20 !       DPDECT        !   TILINE UNIT ERROR COUNT
      +----------+----------+
  >22 !       DPDWTK        !   WORDS PER TRACK
      +----------+----------+
  >24 !  DPDSTK  !  DPDOHD  !   SECTORS PER TRACK
      +----------+----------+      OVERHEAD PER RECORD
  >26 !       DPDCYL        !   HEADS & CYLINDERS
      +----------+----------+
  >28 !  DPDSRD  !  DPDRTK  !   SECTORS PER RECORD
      +----------+----------+      RECORDS PER TRACK
  >2A !       DPDWRD        !   WORDS PER RECORD
      +----------+----------+
  >2C !       DPDILF        !   INTERLEAVING FACTOR
      +----------+----------+
  >2E !       DPDMAD        !   MAX NUMBER OF  ADUS ON DISK
      +----------+----------+
  >30 !       DPDSAD        !   SECTORS PER ADU
      +----------+----------+
  >32 !       DPDDRS        !   DEFAULT PHYSICAL RECORD SIZE
      +----------+----------+
  >34 !       DPDFLG        !   FLAGS
      +----------+----------+
  >36 !  DPDIBF  !          !   INITIALIZATION BUFFER
      +----------+----------+
  >38 !          !          !
      +----------+----------+
  >3A !          !          !
      +----------+----------+
  >3C !       DPDFMS        !   VCAT FD SPECIAL AREA SSB ADDRESS
      +----------+----------+
 >3E !        DPDFDB         !   POINTER TO VCATALOG FCB
     +----------+----------+
 >40 !        DPDPBM         !   DISK MANAGER TABLE/BUFFER ADDR
     +----------+----------+       (NON-ZERO = DISK INSTALLED)
 >42 !  DPDVNM   !           !   VOLUME NAME
     +----------+----------+
     /           /           /
     /           /           /
     +----------+----------+
 >4A !  DPDTFL   !           !   TEMPORARY FILE NAME SEED
     +----------+----------+
     /           /           /
     /           /           /
     +----------+----------+
 >52 !        DPDIVD         !   INSTALLED VOLUME CREATION DATE
     +----------+----------+
 >54 !        DPDIVT         !   INSTALLED VOLUME CREATION TIME
     +----------+----------+


 FLAGS FOR FIELD: DPDFLG    #34 - FLAGS

           = (X...............) -
   DPFRAW = (.X..............) - DISK READ AFTER WRITE
   DPFBRW = (..X.............) - BIT MAP READ AFTER WRITE
   DPFRST = (...X............) - RESTORE FLAG
   DPFSTR = (....X...........) - STORE REGISTER FLAG (1= STORE
*                              REG COMMAND WAS ISSUED BY DSR
*                              TO DETERMINE IF >1B ERROR IS
*                              AN UNSAFE OR MEDIA CHANGE
   DPFODI = (.....X..........) - 1 = ONLINE DIAGNOSTIC REQUEST
   DPFWRP = (......X.........) - SOFTWARE WRITE PROTECT FLAG
   DPFBLF = (.......X........) - BUFFER LOCK FLAG.
*                              1 = THIS DRIVE LOCKED THE
*                                  NON-RE ENTRANT BUFFER USED
*                                  BY MEDIA CHANGE VALIDATION
*                                  PROCESS.
   DPFDTN = (........X.......) - DIRECT TILINE I/O FLAG
           = (.........XXXXXXX) -
```

EQUATES:

LABEL	EQUATE TO	VALUE	DESCRIPTION
DPDSIZ	$	>56	DISK PDT + EXTENSION SIZE

```
***********************************************************
*                                                         *
*      DUTIL DEVICE PARAMETERS    (DPR)     10/04/83      *
*                                                         *
*      CHANGES TO THIS TEMPLATE REQUIRE CORRESPONDING     *
*      CHANGES TO THE PASCAL TEMPLATE "DPRPAS".           *
***********************************************************
*  THE DPR TEMPLATE DESCRIBES THE DEVICE PARAMETERS MANAGED
*  BY THE DEVICE I/O UTILITY (DUTIL).  IT INCLUDES PARAMETERS
*  IN THE FOLLOWING RANGES:
*
*      PARAMETER RANGE       PARAMETER USAGE
*      ---------------       ---------------
*         >01 - >5F          OPERATING SYSTEM RESERVED
*         >60 - >FF          NOT SUPPORTED
*
*  IN THE FIELD COMMENTS, RO INDICATES THAT A PARAMETER IS
*  READ ONLY AND CANNOT BE MODIFIED.
*
*  SPECIAL FIELD COMMENTS:
*  DPRNAM - ONE TO EIGHT ALPHANUMERIC CHARACTERS WITH A LETTER
*           AS THE FIRST CHARACTER.
*  DPRNUM - ONE WORD NUMBER BETWEEN >0001 AND >07FF, EXCLUDING
*           100 THROUGH 255 (>64 THROUGH >FF).
*  DPRTYP - LIKE THE PDTTYP FIELD.  ON AN ASSIGN LUNO, THE VALUE
*           OF THIS FIELD IS PUT INTO THE LDTTYP FIELD OF THE
*           LDT AND IS RETURNED TO THE CALL BLOCK IN THE UPPER
*           BYTE OF THE DATA BUFFER FIELD.
*  DPRJOB - JSB OF THE FIRST JOB TO ASSIGN A LUNO TO A TERMINAL.
*
* EQUATES FOR DPRFLG
*      00 - ONLINE
*      01 - OFFLINE
*      10 - DIAGNOSTIC
*      11 - SPOOLER
* EQUATE FOR DPRDSF
* EQUATES FOR DPRDTF - DEVICE TYPE FLAGS
```

```
*************************************************************
*                                                           *
* DEVICE UTILITY SESSION TABLE   (DUS)           09/09/83 *
*                                                           *
*      LOCATION: IN DUDATA                                  *
*                                                           *
*************************************************************
```

```
    *----------+----------*
>00 !  DUSRES  !  DUSLUN  !   RESERVED AT PRESENT
    +----------+----------+      LUNO OF ACTIVE FILE
>02 !       DUSNAM        !   NAME MANAGER SEGMENT IDS
    +----------+----------+
>04 !                     !
    +----------+----------+
>06 !                     !
    +----------+----------+
>08 !  DUSVOL  !          !   VOLUME NAME OF SYSTEM DISK
    +----------+----------+
    /          /          /
    /          /          /
    +----------+----------+
>10 !  DUSSYS  !          !   SYSTEM NAME
    +----------+----------+
    /          /          /
    /          /          /
    +----------+----------+
>18 !  DUSDTB  !          !   TABLE OF DEVICE TYPE COUNTS
    +----------+----------+
    /          /          /
    /          /          /
    +----------+----------+
```

EQUATES:

LABEL	EQUATE TO	VALUE	DESCRIPTION
MAXSEG	3	>03	MAXIMUM NUMBER OF SEGMENTS
DUSSIZ	$	>26	

```
************************************************************
*                                                          *
*  FILE DESCRIPTOR PACKET     (FDP)               06/22/81 *
*                                                          *
*           LOCATION: ALWAYS A SUB-STRUCTURE               *
************************************************************
* THE FDP IS A TWO WORD ADDRESS OF A FILE CONTROL BLOCK
* (FCB). THE FIRST WORD IS THE SSB ADDRESS OF THE TABLE IN
* WHICH THE SECOND WORD IS THE LOGICAL ADDRESS.


      *----------+----------*
  >00 !        FDPFMT       !   SSB ADDRESS OF FILE MANAGER TABLE
      +----------+----------+
  >02 !        FDPFCB       !   FCB ADDRESS
      +----------+----------+


  EQUATES:

     LABEL   EQUATE TO   VALUE   DESCRIPTION
     -----   ---------   -----   ------------------------------
     FDPSIZ   $             >04
```

```
*************************************************************
*                                                           *
*  FILE DESCRIPTION RECORD    (FDR)             09/09/83    *
*                                                           *
*            LOCATION: DISK                                 *
*************************************************************
* THE FDR IS THE DISK-RESIDENT FILE DESCRIPTOR TELLING WHERE
* THE FILE RESIDES, ITS CHARACTERISTICS, AND SECURITY DATA.
* SECURITY DATA IS STORED IN ACCESS CONTROL ENTRIES (ACEs)
*
                        ** BEGINNING PACKED RECORD ACE


      *----------+----------*
  >00 !  ACEAGN  !          !   ACCESS GROUP NAME
      +----------+----------+
      /          /          /
      /          /          /
      +----------+----------+
  >08 !       ACEFLG        !   FLAGS
      +----------+----------+
  >0A SIZE              ** END OF PACKED RECORD

*


      *----------+----------*
  >00 !       FDRHKC        !   HASH KEY COUNT
      +----------+----------+
  >02 !       FDRHKV        !   HASH KEY VALUE
      +----------+----------+
  >04 !  FDRFNM  !          !   FILE NAME
      +----------+----------+
      /          /          /
      /          /          /
      +----------+----------+
  >0C !  FDRRSV  !          !   RESERVED
      +----------+----------+
  >0E !       FDRFL1        !   FLAGS WORD 1
      +----------+----------+
  >10 !       FDRFLG        !   FLAGS WORD 2
      +----------+----------+
  >12 !       FDRPRS        !   PHYSICAL RECORD SIZE
      +----------+----------+
  >14 !       FDRLRS        !   LOGICAL RECORD SIZE
      +----------+----------+
  >16 !       FDRPAS        !   PRIMARY ALLOCATION SIZE
      +----------+----------+
  >18 !       FDRPAA        !   PRIMARY ALLOCATION ADDRESS
      +----------+----------+
  >1A !       FDRSAS        !   SECONDARY ALLOCATION SIZE
      +----------+----------+
>1C !        FDRSAA         !   OFFSET OF SCONDARY TABLE
    +----------+----------+
>1E !        FDRRFA         !   RECORD NUMBER OF FIRST ALIAS
    +----------+----------+
>20 !  FDREOM  !          !   END OF MEDIUM RECORD NUMBER
    +----------+----------+
>22 !          !          !
    +----------+----------+
>24 !  FDRBKM  !          !   END OF MEDIUM BLOCK NUMBER
    +----------+----------+
>26 !          !          !
    +----------+----------+
>28 !        FDROFM         !   END OF MEDIUM OFFSET/
    +----------+----------+      PRELOG NUMBER FOR KIF
>2A !  FDRFBQ  !          !   FREE BLOCK QUEUE HEAD
    +----------+----------+
>2C !          !          !
    +----------+----------+
>2E !        FDRBTR         !   B-TREE ROOTS BLOCK #
    +----------+----------+
>30 !  FDREBQ  !          !   EMPTY BLOCK QUEUE
    +----------+----------+
>32 !          !          !
    +----------+----------+
>34 !        FDRKDR         !   KEY DESCRIPTIONS RECORD #
    +----------+----------+
>36 !  FDRUD   !          !   LAST UPDATE DATE
    +----------+----------+
>38 !          !          !
    +----------+----------+
>3A !          !          !
    +----------+----------+
>3C !  FDRCD   !          !   CREATION DATE
    +----------+----------+
>3E !          !          !
    +----------+----------+
>40 !          !          !
    +----------+----------+
>42 !  FDRAPB  !  FDRBPA  !   ADU'S PER BLOCK
    +----------+----------+      BLOCKS PER ADU
>44 !        FDRMRS         !   MINIMUM KIF RECORD SIZE
    +----------+----------+
>46 !  FDRSAT  !          !   SECONDARY ALLOCATION TABLE
    +----------+----------+
    /          /          /
    /          /          /
    +----------+----------+
>86 !  FDRRES  !          !   RESERVED
    +----------+----------+
    /          /          /
    /          /          /
    +----------+----------+
>90 !  FDRUID  !          !   USER ID OF FILE CREATOR
```

```
      +----------+----------+
      /          /          /
      /          /          /
      +----------+----------+
 >98  !       FDRPSA        !    PUBLIC SECURITY ATTRIBUTES
      +----------+----------+
 >9A  !       FDRACE        !    9 ACCESS CONTROL ENTRIES
      +----------+----------+
      /          /          /
      /          /          /
      +----------+----------+
 >A4  !                     !
      +----------+----------+
      /          /          /
      /          /          /
      +----------+----------+
 >AE  !                     !
      +----------+----------+
      /          /          /
      /          /          /
      +----------+----------+
 >B8  !                     !
      +----------+----------+
      /          /          /
      /          /          /
      +----------+----------+
 >C2  !                     !
      +----------+----------+
      /          /          /
      /          /          /
      +----------+----------+
 >CC  !                     !
      +----------+----------+
      /          /          /
      /          /          /
      +----------+----------+
 >D6  !                     !
      +----------+----------+
      /          /          /
      /          /          /
      +----------+----------+
 >E0  !                     !
      +----------+----------+
      /          /          /
      /          /          /
      +----------+----------+
 >EA  !                     !
      +----------+----------+
      /          /          /
      /          /          /
      +----------+----------+
 >F4  !  FDRFIL  !          !    NOT USED
      +----------+----------+
     /          /          /
     /          /          /
     +----------+----------+


 FLAGS FOR FIELD: ACEFLG    #08 - FLAGS

   ACERDF = (X...............) - READ ACCESS FLAG
   ACEWRF = (.X..............) - WRITE ACCESS FLAG
   ACEDLF = (..X.............) - DELETE ACCESS FLAG
   ACEEXF = (...X............) - EXECUTE ACCESS FLAG
   ACECTF = (....X...........) - CONTROL ACCESS FLAG


 FLAGS FOR FIELD: FDRFL1    #0E - FLAGS WORD 1

   FDFSEC = (X...............) - FILE SECURED BIT
          = (.XXXXXXXXXXXXXXX) - RESERVED


 FLAGS FOR FIELD: FDRFLG    #10 - FLAGS WORD 2

   FDFFU  = (XX..............) - FILE USAGE BITS
   FDFFMT = (..XX............) - FILE FORMAT BITS
   FDFALL = (....X...........) - EXTENDABLE FILE FLAG
   FDFFT  = (.....XX.........) - FILE TYPE BITS
   FDFWPB = (.......X........) - WRITE PROTECT BIT
   FDFDPB = (........X.......) - DELETE PROTECT BIT
   FDFTMP = (.........X......) - TEMPORARY FILE FLAG
   FDFBLB = (..........X.....) - BLOCKED FILE FLAG
   FDFALI = (...........X....) - ALIAS FLAG BIT
   FDFFWT = (............X...) - FORCED WRITE/PARTIAL LOGGING
          = (.............XX.) - RESERVED
   FDFCDR = (...............X) - RECORD IS CDR
```

EQUATES:

LABEL	EQUATE TO	VALUE	DESCRIPTION
FDFFUM	>C000	>C000	FILE USAGE MASK
FDFFMM	>3000	>3000	FILE FORMAT BITS MASK
FDFFTM	>0600	>600	FILE TYPE MASK
FDFUPM	>0180	>180	WRITE AND DELETE PROTECT MASK
FDRMNT	$	>2A	MAX NUMBER OF TASKS IN PF
FDRMNP	$+1	>2B	MAX NUMBER OF PROCEDURES
FDRMNO	$+2	>2C	MAX NUMBER OF OVERLAYS
FDRSIZ	$	>100	SIZE IN BYTES OF FDR
FDRMAG	9	>09	MAXIMUM ACCESS GROUPS ALLOWED
FDRMNR	5	>05	MAXIMUM # OF RIGHTS DEFINED

```
************************************************************
*                                                          *
*  FILE IDENTIFICATION        (FID)               02/22/80 *
*                                                          *
*            LOCATION:  FILES                              *
************************************************************
* THE FID IS USED TO IDENTIFY WITHIN A FILE ITS NAME AND
* VERSION NUMBER.  THIS IS USED FOR SYSTEM FILES SUCH AS
* S$CLF AND S$SDTQUE.
*
                         ** BEGINNING PACKED RECORD FID


     *----------+----------*
 >00 !  FIDNAM  !          !   FILE NAME
     +----------+----------+
     /          /          /
     /          /          /
     +----------+----------+
 >08 !  FIDVER  !          !   VERSION NUMBER
     +----------+----------+
 >0A !          !          !
     +----------+----------+
 >0C !          !          !
     +----------+----------+
 >0E SIZE                 ** END OF PACKED RECORD
```

```
***********************************************************
*                                                         *
*  FILE INFORMATION RECORD     (FIR)        11/24/82      *
*                                                         *
*            LOCATION: DISK                               *
***********************************************************
* THE FIR IS USED BY THE TASKS WHICH ASSIGN, MODIFY, LIST,
* AND DELETE USER IDS.  IT IS A VARIANT OF THE CAPABILITIES
* LIST FILE RECORD (CLR).  FOR DETAILS SEE CLR.
```

```
************************************************************
*                                                          *
*  FILE STRUCTURE COMMON      (FSC)             02/23/82 *
*                                                          *
*            LOCATION: FILE MANAGEMENT TABLE AREA         *
*                                                          *
************************************************************
* THE FSC IS COMPOSED OF A COMMON FIRST STRUCTURE THAT IS
* SHARED BY BOTH THE FILE CONTROL BLOCK (FCB) AND THE FILE
* DIRECTORY BLOCK (FDB) VARIANTS OF THE REMAINDER OF THE
* STRUCTURE.
*  THE FCB IS AN IN-MEMORY REPRESENTATION THAT IS USED TO
* TRACK THE CHARACTERISTICS OF A FILE THAT IS IN USE. AN FCB
* REPRESENTS THE LAST COMPONENT OF THE FILE PATHNAME.
*  THE FDB IS AN IN-MEMORY STRUCTURE REPRESENTING ONE NODE
* OF THE PATHNAME OF A FILE.  IT PROVIDES TREE LINKAGE FOR
* THE ENTIRE FILE PATHNAME.


      *----------+----------*
  >00 !       FSCPDT        !   PDT POINTER
      +----------+----------+
  >02 !       FSCPDR        !   PTR TO PARENT'S DIRECTORY DOOR
      +----------+----------+
  >04 !       FSCEOM        !   END OF MEDIUM LOGICAL REC #
      +----------+----------+
  >06 !                     !
      +----------+----------+
  >08 !  FSCAPB  !  FSCBPA  !   ADUS PER BLOCK
      +----------+----------+      BLOCKS PER ADU
  >0A !       FSCPAS        !   PRIMARY ALLOCATION SIZE
      +----------+----------+
  >0C !       FSCPAA        !   PRIMARY ALLOCATION ADDRESS
      +----------+----------+
  >0E !       FSCSAA        !   SAT ADDRESS
      +----------+----------+
  >10 !       FSCPRS        !   PHYSICAL RECORD SIZE
      +----------+----------+
  >12 !       FSCADU        !   FDR ADU OF THIS FILE
      +----------+----------+
  >14 !  FSCOFF  !  FSCMFG  !   FDR OFFSET WITHIN ADU
      +----------+----------+      MODIFIED ONLY FLAGS

                          FCB - FILE CONTROL BLOCK VARIANT
      *----------+----------*
  >16 !       FCBFCB        !   LINK FOR CONCATENATED FILES
      +----------+----------+
  >18 !       FCBRPB        !   START OF RPB CHAIN
      +----------+----------+
  >1A !  FCBCCT  !  FCBFLB  !   COUNT OF CONCATENATED FILES
      +----------+----------+      FLAGS BYTE
  >1C !       FCBSGB        !   SGB ADDRESS
```

```
        +----------+----------+
    >1E !        FCBSMT       !   SM TABLE AREA SSB OF SGB
        +----------+----------+
    >20 !        FCBFLG       !   FILE FLAGS
        +----------+----------+
    >22 !        FCBFDB       !   POINTER TO DIRECTORY ENTRY
        +----------+----------+
    >24 !        FCBSFD       !   SSB OF DIRECTORY ENTRY
        +----------+----------+
    >26 !        FCBLRS       !   LOGICAL RECORD SIZE
        +----------+----------+
    >28 !        FCBSAS       !   SECONDARY ALLOCATION SIZE
        +----------+----------+
    >2A !        FCBBKM       !   END OF MEDIUM BLOCK #
        +----------+----------+
    >2C !                     !
        +----------+----------+
    >2E !        FCBOFM       !   END OF MEDIUM OFFSET
        +----------+----------+
    >30 !        FCBLRL       !   LOCKED RECORD LIST HEAD
        +----------+----------+
    >32 !        FCBEXT       !   BLOCK COUNT FOR FILE EXTENT
        +----------+----------+
    >34 !                     !
        +----------+----------+
    >36 !  FCBXCT  !  FCBCLA  !   FILE EXTENSION COUNT
        +----------+----------+      COUNT OF THINGS POINTING HERE
    >38 !        FCBRLA       !   REQUEST LIST ANCHOR
        +----------+----------+
    >3A !  FCBCPO  !  FCBCAW  !   COUNT OF PASSIVE OPERATIONS
        +----------+----------+      COUNT OF ACTIVE WAITERS


        *----------+----------*
    >3C !        FCBEBQ       !   EMPTY BLOCK QUEUE
        +----------+----------+
    >3E !                     !
        +----------+----------+
    >40 !        FCBCLB       !   CURRENT LOG BLOCK #
        +----------+----------+
    >42 !  FCBFBQ  !          !   FREE BLOCK QUEUE HEAD
        +----------+----------+
    >44 !          !          !
        +----------+----------+
    >46 !        FCBBTR       !   B-TREE ROOTS BLOCK #
        +----------+----------+
    >48 !        FCBSBB       !   STARTING BUCKET BLOCK #
        +----------+----------+
    >4A !        FCBMRS       !   MINIMUM KIF RECORD SIZE
        +----------+----------+
    >4C !  FCBKDB  !          !   KEY DESCRIPTIONS BLOCK
        +----------+----------+
        /          /          /
    /          /          /
    +----------+----------+


    *----------+----------*
>3C !  FCBMNT  !  FCBTO   !   MAXIMUM NUMBER OF TASKS
    +----------+----------+      TASK DIRECTORY ENTRY OFFSET
>3E !       FCBTR         !   TASK DIRECTORY ENTRY RECORD #
    +----------+----------+
>40 !  FCBMNP  !  FCBPO   !   MAXIMUM NUMBER OF PROCEDURES
    +----------+----------+      PROC DIRECTORY ENTRY OFFSET
>42 !       FCBPR         !   PROC DIRECTORY ENTRY RECORD #
    +----------+----------+
>44 !  FCBMNO  !  FCBOO   !   MAXIMUM NUMBER OF OVERLAYS
    +----------+----------+      OVERLAY DIRECTORY ENTRY OFFSET
>46 !       FCBOR         !   OVERLAY DIRECTORY ENTRY RECORD
    +----------+----------+

                          FDB - FILE DIRECTORY BLOCK VARIANT
    *----------+----------*
>16 !       FDBRNM        !   RECORD NUMBER OF FDR
    +----------+----------+
>18 !       FDBDDR        !   ADDRESS OF DIRECTORY DOOR (DDR)
    +----------+----------+
>1A !  FDBFNM  !          !   FILE NAME
    +----------+----------+
    /          /          /
    /          /          /
    +----------+----------+
>22 !       FDBFCB        !   ADDRESS OF FCB ANCHOR
    +----------+----------+
>24 !  FDBCDF  !  FILL02  !   COUNT OF DESCENDANTS
    +----------+----------+      RESERVED
>26 !       FDBAFD        !   ADDRESS OF FIRST DESCENDANT (AFD)
    +----------+----------+
>28 !       FDBSAF        !   SSB ADDRESS FOR AFD
    +----------+----------+
>2A !       FDBALS        !   ADDRESS OF LAST SIBLING (ALS)
    +----------+----------+
>2C !       FDBSAL        !   SSB ADDRESS FOR ALS
    +----------+----------+
>2E !       FDBANS        !   ADDRESS OF NEXT SIBLING (ANS)
    +----------+----------+
>30 !       FDBSAN        !   SSB ADDRESS FOR ANS
    +----------+----------+
>32 !       FDBAPF        !   ADDRESS OF PARENT FILE (APF)
    +----------+----------+
>34 !       FDBSAP        !   SSB ADDRESS FOR APF
    +----------+----------+
>36 !       FDBFMT        !   SSB ADDRESS FOR THIS FDB
    +----------+----------+
  FLAGS FOR FIELD: FSCMFG    #15 - MODIFIED ONLY FLAGS

    FSCMEC = (X...............) - 1 = END OF MEDIUM HAS CHANGED
    FSCMWT = (.X..............) - 1 = FILE HAS BEEN WRITTEN IN
    FSCFU1 = (..XX............) - FILE USAGE BIT ONE
    FSCDEL = (....X...........) - FDB DELETE PROTECTION FLAG
*
*


  FLAGS FOR FIELD: FCBFLB    #1B - FLAGS BYTE

    FCBFCC = (X...............) - FILE IS IN CONCATENATION
    FCBFUB = (.X..............) - OPEN MUST BE UNBLOCKED
    FCBBSY = (..X.............) - FCB IS BUSY
    FCBFSE = (...X............) - SUPPRESS EOF BEFORE EOM


  FLAGS FOR FIELD: FCBFLG    #20 - FILE FLAGS

    FCBFFU = (XX..............) - FILE USAGE FLAGS
*                                 00 = NO SPECIAL USAGE
*                                 01 = DIRECTORY
*                                 10 = PROGRAM
*                                 11 = IMAGE
    FCBFDF = (..XX............) - DATA FORMAT
*                                 00 = NON-BLANK SUPPRESSED
*                                 01 = BLANK SUPPRESSED
*                                 10 & 11 = RESERVED
    FCBFAT = (....X...........) - EXPANDABLE IF ON
    FCBFFT = (.....XX.........) - FILE TYPE
*                                 00 = RESERVED (FOR DEVICE)
*                                 01 = SEQUENTIAL
*                                 10 = RELATIVE RECORD
*                                 11 = KEY INDEXED
    FCBFWP = (.......X........) - WRITE PROTECTED IF ON
    FCBFDP = (........X.......) - DELETE PROTECTED IF ON
    FCBFTF = (.........X......) - TEMPORARY FILE IF ON
    FCBFBF = (..........X.....) - BLOCKED FILE IF OFF
    FCBFAF = (...........X....) - ALIAS ENTRY IF ON
    FCBFFW = (............X...) - FORCED WRITE IF ON
    FCBFSC = (.............X..) - FILE SECURITY
    FCBPLG = (..............X.) - FILE MARKED AS PARTIAL LOGGING
    RESER  = (...............X) - RESERVED


  EQUATES:

     LABEL   EQUATE TO   VALUE   DESCRIPTION
     -----   ---------   -----   ------------------------------
     FSCFU2   FSCFU1+1     >03    FILE USAGE BIT TWO
     FSCFUM   >3000        >3000    FILE USAGE MASK
     FSCSIZ   $            >16    FSC SIZE
```

FCBFFM	>C000	>C000	FILE USAGE FLAGS MASK
FCBFAM	>3000	>3000	FILE FORMAT FLAGS MASK
FCBFTM	>0600	>600	FILE TYPE FLAGS MASK
FCBMSZ	$	>3C	MIN FCB SIZE
FCBSIZ	$	>86	MAX FCB SIZE
FCBPSZ	$	>48	PROGRAM FILE FCB SIZE
FCBPDT	FSCPDT	>00	PDT ADDRESS
FCBPDR	FSCPDR	>02	PTR TO PARENT'S DIRECTORY DOOR
FCBEOM	FSCEOM	>04	END OF MEDIUM LOGICAL REC. #
FCBAPB	FSCAPB	>08	ADUS PER BLOCK
FCBBPA	FSCBPA	>09	BLOCKS PER ADU
FCBPAS	FSCPAS	>0A	PRIMARY ALLOCATION SIZE
FCBPAA	FSCPAA	>0C	PRIMARY ALLOCATION ADDRESS
FCBSAA	FSCSAA	>0E	SAT ADDRESS
FCBPRS	FSCPRS	>10	PHYSICAL RECORD SIZE
FCBADU	FSCADU	>12	FDR ADU FOR THIS FILE
FCBOFF	FSCOFF	>14	FDR OFFSET WITHIN ADU
FCBMFG	FSCMFG	>15	MODIFIED ONLY FLAGS
FCBMEC	FSCMEC	>00	1= EOM HAS CHANGED
FCBMWT	FSCMWT	>01	1= FILE WAS WRITTEN IN
FDBMSZ	$	>38	FDB SIZE
FDBPDT	FSCPDT	>00	PDT ADDRESS
FDBPDR	FSCPDR	>02	PTR TO PARENT'S DIRECTORY DOOR
FDBEOM	FSCEOM	>04	END OF MEDIUM LOGICAL REC. #
FDBAPB	FSCAPB	>08	ADUS PER BLOCK
FDBBPA	FSCBPA	>09	BLOCKS PER ADU
FDBPAS	FSCPAS	>0A	PRIMARY ALLOCATION SIZE
FDBPAA	FSCPAA	>0C	PRIMARY ALLOCATION ADDRESS
FDBSAA	FSCSAA	>0E	SAT ADDRESS
FDBPRS	FSCPRS	>10	PHYSICAL RECORD SIZE
FDBADU	FSCADU	>12	FDR ADU FOR THIS FILE
FDBOFF	FSCOFF	>14	FDR SECTOR OFFSET IN ADU
FDBMFG	FSCMFG	>15	MODIFIED ONLY FLAGS
FDBMEC	FSCMEC	>00	1 = EOM HAS CHANGED
FDBMWT	FSCMWT	>01	1 = FILE WAS WRITTEN IN
FDBFU1	FSCFU1	>02	FILE USAGE BIT ONE
FDBFU2	FSCFU2	>03	FILE USAGE BIT TWO
FDBFDL	FSCDEL	>04	FDB DELETE PROTECTION FLAG

```
************************************************************
*                                                          *
*  FILE MANAGER WORK AREA     (FWA)             01/21/82   *
*                                                          *
*             LOCATION: SYSTEM AREA                        *
************************************************************
*  THE FWA IS USED BY FILE MANAGEMENT AND BY KIF MANAGEMENT
*  AS A GENERAL WORK AREA.  R15 POINTS TO THE FWA.


      *----------+----------*
  >00 !  FWAWP   !          !   WORKSPACE USED BY FM
      +----------+----------+
      /          /          /
      /          /          /
      +----------+----------+
  >20 !       FWAFLG        !   MIDDLE SEGMENT FLAGS
      +----------+----------+
  >22 !       FWATCT        !   MULTIRECORD CHARS TRANSFERRED
      +----------+----------+
  >24 !       FWAOAD        !   CURRENT OVERLAY AREA ADDRESS
      +----------+----------+
  >26 !       FWAPC         !   SAVED PROGRAM COUNTER
      +----------+----------+
  >28 !       FWAXWP        !   BLWP VECTOR FOR RETURNING
      +----------+----------+
  >2A !       FWAXPC        !
      +----------+----------+
  >2C !       FWABN         !   SAVED RPBBN (2 WORDS)
      +----------+----------+
  >2E !                     !
      +----------+----------+
  >30 !       FWAOCB        !   SAVED RPBOCB
      +----------+----------+
  >32 !       FWALFG        !   SAVED LDT FLAGS
      +----------+----------+
  >34 !       FWAFFC        !   FIRST FCB FOR CC FILES
      +----------+----------+
  >36 !       FWAFMT        !   SSB FOR THE FMT WITH THE FCB
      +----------+----------+
  >38 !       FWAFCB        !   FCB ADDRESS IN THE FMT
      +----------+----------+
  >3A !       FWABST        !   SMT SSB ADDR FOR BUFFER
      +----------+----------+
  >3C !       FWABSB        !   SSB ADDR FOR BUFFER
      +----------+----------+
  >3E !       FWAPRS        !   PHYSICAL RECORD SIZE
      +----------+----------+
  >40 !       FWAUBT        !   USER BUFFER SMT SSB ADDR
      +----------+----------+
  >42 !       FWAUBS        !   USER BUFFER SSB ADDRESS
      +----------+----------+
>44 !         FWAUBO        !   USER BUFFER OFFSET
    +----------+----------+
>46 !         FWAUBL        !   USER BUFFER LENGTH
    +----------+----------+
>48 !         FWAFMB        !   FMT BIAS
    +----------+----------+
>4A !         FWARN1        !   RECORD # RELATIVE TO CURRENT
    +----------+----------+
>4C !         FWARN2        !   FILE OF CONCATENATED SET
    +----------+----------+
>4E !         FWAOOB        !   OLD OFFSET IN USER BUFFER
    +----------+----------+
>50 !         FWAUBR        !   BUFFER LENGTH REMAINING
    +----------+----------+
>52 !         FWAFFG        !   FILE MGR FLAGS
    +----------+----------+
>54 !  FWASTK  !          !   STACK AREA
    +----------+----------+
    /          /          /
    /          /          /
    +----------+----------+
```

```
FLAGS FOR FIELD: FWAFFG    #52 - FILE MGR FLAGS

  FWAPOP = (X...............) - PASSIVE OPERATION FLAG
  FWAQW  = (.X..............) - QUEUED TO WAITING QUEUE IN FCB
```

```
EQUATES:

   LABEL   EQUATE TO   VALUE   DESCRIPTION
   -----   ---------   -----   ------------------------------
   FWASIZ  $             >F4    SIZE OF FWA INCLUDING WSP
```

```
***********************************************************
*                                                         *
*  I/O REQUEST BLOCK     (IRB)                   09/09/83*
*                                                         *
*           LOCATION: SYSTEM TABLE AREA AND JCA           *
*                                                         *
***********************************************************
* THE IRB TEMPLATE HAS FOUR MAJOR VARIANTS.  ONE OF THESE IS
* THE SIMPLE CALL BLOCK FOR RESOURCE INDEPENDENT I/O. ONE
* HAS EXTENSIONS FOR VDT DEVICES.  ANOTHER IS THE CALL
* BLOCK USED FOR I/O UTILITY CALLS.  IT INCLUDES INTERNAL
* VARIANTS FOR REMOTE I/O HANDLING AND FOR LOGICAL NAME
* SEGMENT HANDLING.  THERE IS ALSO A SET OF EQUATES USED BY
* THE CODE WHICH CREATES PROGRAM FILES.  EQUATES FOR SPECIAL
* PURPOSES IN CREATING KEY INDEXED FILES AND FOR REFERENCE
* TO SPECIAL APPLICATIONS OF THE BASIC I/O BLOCK ARE IMBEDDED
* IN THE TEMPLATE WHERE THE ORIGINAL FIELDS ARE DEFINED.
* A FINAL VARIANT IS USED FOR FILE I/O CALL BLOCKS.
*------
* NOTE THAT FOR DUPLICATE LABELS, THE PREFERRED USAGE IS
* STARRED IN THE COMMENT COLUMNS.
                              ** BEGINNING PACKED RECORD IRB


      *----------+----------*
  >00 !  IRBSOC  !  IRBEC   !   SUPERVISOR REQUEST CODE
      +----------+----------+      *REQUEST ERROR CODE
  >02 !  IRBOC   !  IRBLUN  !   *SUB-OPERATION CODE
      +----------+----------+      LOGICAL UNIT
  >04 !  IRBSFL  !  IRBUFL  !   *SYSTEM FLAGS
      +----------+----------+      REQUESTOR (USER) FLAGS

                                RESOURCE-INDEPENDENT I/O VARIANT
      *----------+----------*
  >06 !        IRBDBA       !   *DATA BUFFER ADDRESS
      +----------+----------+
  >08 !        IRBICC       !   *INPUT CHAR COUNT / ACTUAL OUTPUT
      +----------+----------+
  >0A !        IRBOCC       !   *OUTPUT CHAR COUNT / ACTUAL INPUT
      +----------+----------+

                                FILE I/O VARIANTS
      *----------+----------*
  >0C !        IRBCBA       !   CURRENCY BLOCK ADDRESS
      +----------+----------+


      *----------+----------*
  >0C !        IRBRN1       !   RELATIVE RECORD NUMBER
      +----------+----------+
  >0E !                     !
      +----------+----------+
>10 !         FILL01        !
    +----------+----------+

                           DIAGNOSTIC PORT VARIANT
    *----------+----------*
>0C !  IRBVRS  !  IRBOLF  !    OS VERSION/RELEASE
    +----------+----------+       ONLINE FLAGS
>0E !         IRBPCD      !    DYNAMIC PASSCODE
    +----------+----------+

                           DIRECT DISK I/O
    *----------+----------*
>0C !         IRBADU      !    ADU ADDRESS
    +----------+----------+
>0E !         IRBOFF      !    SECTOR OFFSET
    +----------+----------+


    *----------+----------*
>0C !         IRBTRK      !    TRACK ADDRESS
    +----------+----------+
>0E !  IRBSPR  !  IRBSCT  !    SECTORS PER RECORD
    +----------+----------+       SECTOR

                           SUBOPCODE >18 VARIANT
    *----------+----------*
>0C !         IRBDKS      !    TPCS (R0) DISK STATUS FOR >18
    +----------+----------+
>0E !         IRBCRS      !    TPCS (R7) CONTROLLER STAT FOR >18
    +----------+----------+

                           TERMINAL DEVICE I/O VARIANTS
    *----------+----------*
>0C !         IRBRPY      !    REPLY BLOCK ADDRESS
    +----------+----------+
>0E !         IRBXFL      !    EXTENDED REQUEST FLAGS
    +----------+----------+
>10 !  IRBFCH  !  IRBEVT  !    VDT FILL CHARACTER
    +----------+----------+       VDT EVENT BYTE
>12 !  IRBCRO  !  IRBCCO  !    VDT CURSOR IN FIELD ROW
    +----------+----------+       VDT CURSOR IN FIELD COLUMN
>14 !  IRBFRO  !  IRBFCO  !    VDT FIELD BEGINNING ROW
    +----------+----------+       VDT FIELD BEGINNING COLUMN


    *----------+----------*
>0C !         IRBVTA      !    VALIDATION TABLE ADDRESS
    +----------+----------+

                           I/O UTILITY VARIANT
    *----------+----------*
>06 !  IRBTYP  !  IRBTFL  !    RESOURCE TYPE
    +----------+----------+       RESOURCE TYPE FLAGS
```

```
>08 !        IRBJSB        !   OWNER JSB ADDRESS (IOU SVC >A5)
    +----------+----------+
>0A !        IRBTSB        !   OWNER TSB ADDRESS (IOU SVC >A5)
    +----------+----------+
>0C !        IRBKDB        !   KEY DESCRIPTOR ADDRESS/OVERLAYS IRBA
    +----------+----------+
>0E !        FILL04        !   OVERLAYS IRBOFF
    +----------+----------+
>10 !        IRBFLG        !   UTILITY FLAGS (2 BYTES)
    +----------+----------+
>12 !        IRBDLL        !   DEFINED LOGICAL  RECORD LENGTH
    +----------+----------+
>14 !        IRBDPL        !   DEFINED PHYSICAL RECORD LENGTH
    +----------+----------+
>16 !        IRBPNA        !   PATHNAME ADDRESS
    +----------+----------+
>18 !        IRBPRM        !   PARAMETER POINTER
    +----------+----------+
>1A !        IRBRES        !   RESERVED
    +----------+----------+
>1C !        IRBIFA        !   INITIAL FILE ALLOCATION (2 WORDS)
    +----------+----------+
>1E !                      !
    +----------+----------+
>20 !        IRBSFA        !   SECONDARY FILE ALLOCATION (2 WORDS)
    +----------+----------+
>22 !                      !
    +----------+----------+

                          IOU VARIANT FOR LOGICAL NAME SEGMENT
    *----------+----------*
>24 !  IRBSTG  !  IRBNMF  !   TASK STAGE NUMBER
    +----------+----------+      NAME MANAGER FLAGS
>26 !        IRBRPN        !   REDIRECTED RESOLVED PATHNAME
    +----------+----------+      EQUATES FOR CREATE PROGRAM FILE VAR
>28 SIZE                ** END OF PACKED RECORD


FLAGS FOR FIELD: IRBSFL    #04 - *SYSTEM FLAGS

  IRFBSY = (X...............) - BUSY
  IRFERR = (.X..............) - ERROR
  IRFEOF = (..X.............) - END OF FILE
  IRFVNT = (...X............) - EVENT CHAR
         = (....XXX.........) -
  IRFMDT = (.......X........) - MODIFIED DATA TAG (OPCODE >17)


FLAGS FOR FIELD: IRBUFL    #05 - REQUESTOR (USER) FLAGS

  IRFINT = (X...............) - INITIATE REQUEST
  IRFRPY = (.X..............) - OUTPUT WITH REPLY
```

```
    IRFSAR = (..X.............) - SECURITY ACCESS RIGHTS (OPCODE >5)
    IRFACC = (...XX...........) - ACCESS PRIVILEGES
*                                 00=EXCLUSIVE WRITE
*                                 01=EXCLUSIVE ALL
*                                 10=SHARED
*                                 11=READ ONLY
    IRFLOC = (.....X..........) - *LOCK/UNLOCK
    IRFOWN = (......XX........) - OWNERSHIP LEVEL


  FLAGS FOR FIELD: IRBOLF    #0D - ONLINE FLAGS

    OLDBFR = (X...............) - BUFFER ADDRESS SPECIFIED
*                               0=NO BUFFER IN TIL. IMAGE
*                               1=BUFFER IN TILINE IMAGE
           = (.XXXXXXX........) - RESERVED (SET TO 0)


  FLAGS FOR FIELD: IRBXFL    #0E - EXTENDED REQUEST FLAGS

    IRFCSF = (X...............) - CURSOR START OF FIELD DEFN
    IRFNTN = (.X..............) - INTENSITY
    IRFFKR = (..X.............) - BLINKING CURSOR (FLICKER)
    IRFGRA = (...X............) - GRAPHICS DISPLAY(CHAR LT >20)
    IRFEBA = (....X...........) - 8-BIT ASCII
    IRFTER = (.....X..........) - ENABLE TASK EDIT CHAR RETURN
    IRFBP  = (......X.........) - BEEP
    IRFRDB = (.......X........) - RIGHT DISPLAY EDGE BOUNDARY
    IRFCIF = (........X.......) - CURSOR IN-FIELD DEFINED
    IRFFC  = (.........X......) - FILL CHAR DEFINED
    IRFIF  = (..........X.....) - INITIALIZE FIELD
    IRFRFF = (...........X....) - REMAIN IN FULL FIELD
    IRFECO = (............X...) - ECHO
    IRFVRQ = (.............X..) - VALIDATION REQUIRED
    IRFVER = (..............X.) - VERIFICATION ERROR
    IRFWBP = (...............X) - WARNING BEEP


  FLAGS FOR FIELD: IRBTFL    #07 - RESOURCE TYPE FLAGS

           = (XXX.............) - RESERVED
    IRFVD  = (...X............) - VIRTUAL DEVICE
    IRFREM = (....X...........) - REMOTE CHANNEL
    IRFCHN = (.....X..........) - CHANNEL
    IRFDEV = (......X.........) - DEVICE
    IRFFIL = (.......X........) - FILE


  FLAGS FOR FIELD: IRBFLG    #10 - UTILITY FLAGS (2 BYTES)

    IRFFCA = (X...............) - FILE CREATED BY ASSIGN
    IRFFU1 = (.XX.............) - FILE USAGE FLAGS
*                                 00=NO SPECIAL USAGE
*                                       01=DIRECTORY FILE
*                                       10=PROGRAM FILE
*                                       11=IMAGE FILE
    IRFSC1 = (...XX...........) - LUNO SCOPE
*                                       00=TASK LOCAL
*                                       01=JOB LOCAL
*                                       10=GLOBAL
*                                       11=SHARED
    IRFGEN = (.....X..........) - AUTOGENERATE LUNO
    IRFACR = (......X.........) - REQUEST AUTOCREATE FILE
    IRFPRM = (.......X........) - 1=IRBPRM VALID (PARMS PRESENT)
    IRFLRL = (........X.......) - 1=VALID LOGICAL RECORD LENGTH
    IRFTMP = (.........X......) - FILE IS TO BE TEMPORARY
    IRFIMW = (..........X.....) - IMMEDIATE WRITE DISK FILES
    IRFDF1 = (...........XX...) - DATA FORMAT
*                                       00=NORMAL RECORD IMAGE
*                                       01=BLANK SUPPRESSED
*                                       10,11 RESERVED
    IRFALL = (.............X..) - ALLOCATION MAY GROW
    IRFFT1 = (..............XX) - FILE TYPE
*                                       00=RESERVED
*                                       01=SEQUENTIAL FILE
*                                       10=RELATIVE RECORD FILE
*                                       11=KEY INDEXED FILE


  FLAGS FOR FIELD: IRBNMF     #25 - NAME MANAGER FLAGS

    IRFRID = (X...............) - USE SPECIFIED RUN ID
    IRFNM1 = (.XXXXXXX........) - RESERVED AT PRESENT
```

EQUATES:

LABEL	EQUATE TO	VALUE	DESCRIPTION
IRBERR	IRBEC	>01	REQUEST ERROR CODE
IRBOP	IRBOC	>02	SUB-OPERATION CODE
IRBSFG	IRBSFL	>04	SYSTEM FLAGS
IRFVAL	IRFRPY	>01	READ WITH VALIDATION
IRFKFG	IRFRPY	>01	KEY SPECIFIES FLAG
IRFBFI	IRFRPY	>01	BUFF HAS WRITE INTERLEAVED FMT
IRFRES	IRFSAR	>02	
IRFAC1	IRFACC+1	>04	ACCESS PRIVILEGES
IRFRLN	IRFACC	>03	MASTER RESOLVE LOGICAL NAMES
IRFACM	>0018	>18	ACCESS PRIV. BIT MASK
IRFOS	IRFACC	>03	READ BY TRACK/OFFSET ENABLED
IRFOSF	IRFACC+1	>04	READ BY TRACK/OFFSET FORWARD
IRFMDS	IRFLOC	>05	MASTER DO NOT SUSPEND
IRFLFG	IRFLOC	>05	LOCK/UNLOCK
IRFTIH	IRFLOC	>05	READ BY TRACK/TRANSFER INHIBIT
IRFIMO	IRFLOC	>05	IMMEDIATE OPEN FLAG FOR TPD
IRFPAS	IRFLOC	>05	PASS THRU MODE FOR COMM DSRS

```
IRFEXR    IRFOWN        >06     EXTENDED REQUEST
IRFBAD    IRFOWN+1      >07     *BLANK ADJ/SET EVENT MODE
IRFBFG    IRFBAD        >07     BLANK ADJ/SET EVENT MODE
IRFWPM    IRFOWN+1      >07     WORD PROCESSING MODE
IRFRTY    IRFOWN+1      >07     READ BY TRACK WITH NO RETRIES
VARNT1    $             >06
IRBRLN    IRBICC        >08     RECORD LENGTH
IRBLRL    IRBICC        >08     LOGICAL RECORD LENGTH
IRBCHT    IRBOCC        >0A     OUTPUT CHARACTER COUNT
IRBCMD    IRBOCC        >0A     COMMAND FOR SUBOPCODE >18
IRBHD     IRBOCC+1      >0B     HEAD # FOR SUBOPCODE >18
VARNT2    $             >0C
IRBCYL    IRBRN1        >0C     CYLINDER ADDRESS FOR SUBOP >18
IRBRN2    IRBRN1+2      >0E     SECOND WORD OF RECORD NUMBER
IRBFWA    $             >10     POINTER TO FILE WORK AREA
IRBRT     IRBXFL        >0E     RESOURCE TYPE/TYPE FLAGS FOR CC
IRBLRN    IRBKDB        >0C     LOGICAL RECORD NUMBER
IRBPRS    IRBKDB        >0C     PHYSICAL RECORD SIZE (DIR OVHD.)
IRFFU2    IRFFU1+1      >02     FILE USAGE FLAGS
IRFFUM    >6000         >6000     FILE USAGE BIT MASK
IRFSC2    IRFSC1+1      >04     LUNO SCOPE
IRFSCM    >1800         >1800     LUNO SCOPE BIT MASK
IRFDF2    IRFDF1+1      >0C     DATA FORMAT
IRFDFM    >0018         >18     DATA FORMAT BIT MASK
IRFFT2    IRFFT1+1      >0F     FILE TYPE
IRFFTM    >0003         >03     FILE TYPE BIT MASK
IRBIF2    IRBIFA+2      >1E     INITIAL FILE ALLOCATION (2ND WORD)
IRBRCS    $             >24     REQUESTOR CALL BLOCK SIZE
VARNT3    $             >24
IRBMXT    IRBRPY+1      >0D     MAXIMUM NUMBER TASKS IN PROG FILE
IRBMXP    IRBXFL        >0E     MAX NUMBER PROCS IN PROG FILE
IRBMXO    IRBXFL+1      >0F     MAX NUMBER OVERLAYS IN PROG FILE
```

```
*************************************************************
*                                                           *
*  JOB INFORMATION TABLE        (JIT)        01/23/80       *
*                                                           *
*           LOCATION: JCA                                   *
*************************************************************
* THE JIT DESCRIBES THE JOB COMMUNICATION AREA (JCA) CONTENTS
* AND IS FOUND AT THE ADDRESS JCASTR (FOUND IN NFPTR) IN
* EACH JOB.  IT INCLUDES DESCRIPTIVE INFORMATION ABOUT THE
* JOB AND POINTERS TO MANY JOB-LOCAL STRUCTURES IN THE JCA.
* (NOTE THAT JITOVB MUST BE ON A BEET BOUNDARY.)


      *----------+----------*
  >00 !       JITHED        !    SYSTEM TABLE AREA OVERHEAD
      +----------+----------+
  >02 !       JITLNK        !    STA - LINK TO FIRST BLOCK
      +----------+----------+
  >04 !       JITRES        !    STA - RESERVE LIMIT
      +----------+----------+
  >06 !       JITEND        !    STA - END OF AREA
      +----------+----------+
  >08 !       JITUSE        !    STA - TOTAL BYTES USED
      +----------+----------+
  >0A !       JITHI         !    STA - HIGHEST ADDR USED
      +----------+----------+
  >0C !       JITPTR        !    POINTER TO JSB OF JCA OWNER
      +----------+----------+
  >0E !       JITCAP        !    POINTER TO CAPABILITY LIST
      +----------+----------+
  >10 !       JITSEM        !    POINTER TO SEMAPHORE LIST
      +----------+----------+
  >12 !       JITLDT        !    JOB LOCAL LDT - LDT LINK
      +----------+----------+
  >14 !  JITIOC  !  JITLUN  !    LDT - INITIATE I/O COUNT
      +----------+----------+       LDT - LUNO
  >16 !  JITTYP  !  JITTF   !    LDT - RESOURCE TYPE
      +----------+----------+       LDT - RESOURCE FLAGS
  >18 !       JITFLG        !    LDT - LDT FLAGS
      +----------+----------+
  >1A !       JITRLK        !    LDT - RESOURCE LINK
      +----------+----------+
  >1C !       JITOTS        !    LDT - OWNER TSB
      +----------+----------+
  >1E !       JITJSB        !    LDT - OWNER JSB
      +----------+----------+
  >20 !       JITPRM        !    LDT - PARAMETER LIST
      +----------+----------+
  >22 !       JITROB        !    POINTER TO RESOURCE OWNER BLK
      +----------+----------+
  >24 !       JITCCB        !    POINTER TO CHANNEL CONTROL BLK
      +----------+----------+
>26 !  JITPAS   !          !   PASSWORD FOR USER ID
    +----------+----------+
    /          /          /
    /          /          /
    +----------+----------+
>2E !  JITACC   !          !   ACCOUNT ID
    +----------+----------+
    /          /          /
    /          /          /
    +----------+----------+
>3E !  JITPVL   !  JITLID  !   USER PRIVILEGE LEVEL
    +----------+----------+      LAST TASK ID GIVEN
>40 !  JITTID   !          !   TSB RUN TIME ID BIT MAPS
    +----------+----------+
    /          /          /
    /          /          /
    +----------+----------+
>60 !        JITJBI        !   SEGMENT ID OF JCA
    +----------+----------+
>62 !        JITFTB        !   POINTER TO FM TSB
    +----------+----------+
>64 !        JITFLK        !   FORWARD TOL LINK FOR FM
    +----------+----------+
>66 !        JITBLK        !   BACKWARD TOL LINK FOR FM
    +----------+----------+
>68 !        JITFTP        !   FM TASK TYPE (>0100)
    +----------+----------+
>6A !        JITFJB        !   POINTER TO FM JSB
    +----------+----------+
>6C !  FILL00   !          !   FILLER FOR DUMY OVB
    +----------+----------+
    /          /          /
    /          /          /
    +----------+----------+
>80 !  JITFMQ   !          !   FM QUEUE FOR JOB
    +----------+----------+
    /          /          /
    /          /          /
    +----------+----------+
>8C !        JITEXC        !   EXECUTION TIME SINCE LOAD
    +----------+----------+
>8E !                      !
    +----------+----------+
>90 !        FILL01        !   CURRENTLY UNUSED
    +----------+----------+
>92 !        JITWOT        !   TABLE AREA WAIT QUEUE
    +----------+----------+
>94 !        FILL02        !   RESERVED AT PRESENT
    +----------+----------+
>96 !        JITSSI        !   SYNONYM SEGMENT RUN ID
    +----------+----------+
>98 !        JITSSS        !   SYNONYM SEGMGR TAB AREA SSB
    +----------+----------+
  >9A !        JITSSB         !    SYNONYM SSB ADDRESS
      +-----------+-----------+
  >9C !        JITRST         !    PTR TO RESERVE SEGMENT TABLE
      +-----------+-----------+
  >9E !        JITTSB         !    POINTER TO TSB TREE
      +-----------+-----------+
  >A0 !        JITACT         !    POINTER TO ACTIVE TSBS
      +-----------+-----------+
  >A2 !        JITWOM         !    POINTER TO WOM TSBS
      +-----------+-----------+
  >A4 !  JITFMW   !           !    FILE MGR WORKING WP & STACK
      +-----------+-----------+
      /           /           /
      /           /           /
      +-----------+-----------+
  >E2 !  JITINS   !           !    INSTALL TASK QUEUE HEADER
      +-----------+-----------+
      /           /           /
      /           /           /
      +-----------+-----------+
  >EE !  JITDEL   !           !    DELETE TASK QUEUE HEADER
      +-----------+-----------+
      /           /           /
      /           /           /
      +-----------+-----------+
  >FA !  JITASP   !           !    ASSIGN SPACE QUEUE HEADER
      +-----------+-----------+
      /           /           /
      /           /           /
      +-----------+-----------+
>0106 !  JITMAP   !           !    MAP NAME TO ID QUEUE HEADER
      +-----------+-----------+
      /           /           /
      /           /           /
      +-----------+-----------+
>0112 !  JITINV   !           !    INITIALIZE NEW VOLUME QUEUE
      +-----------+-----------+
      /           /           /
      /           /           /
      +-----------+-----------+
>011E !  JITRCP   !           !    RETURN CODE PROCESSOR QUEUE
      +-----------+-----------+
      /           /           /
      /           /           /
      +-----------+-----------+
```

EQUATES:

LABEL	EQUATE TO	VALUE	DESCRIPTION
JITOVB	$	>60	
JITSIZ	$	>12A	

```
**********************************************************
*                                                        *
*  JOB MANAGEMENT REQUEST    (JMR)             04/28/79 *
*            LOCATION:  SYSTEM TABLE AREA               *
*                                                        *
**********************************************************
* THE JMR IS A DESCRIPTION OF A JOB MANAGEMENT SVC BLOCK.
* IT IS USED WITHIN JOB MANAGMENT TO SCAN THE USER'S SVC
* REQUEST.


     *----------+----------*
 >00 !  JMRSVC  !  JMRERR  !   SVC CODE  (48)
     +----------+----------+      ERROR CODE
 >02 !  JMROP   !  JMRPRI  !   JOB MANAGER SUBOPCODE
     +----------+----------+      JOB PRIORITY
 >04 !       JMRFLG        !   JOB MANAGER CONTROL FLAGS
     +----------+----------+
 >06 !       JMRJID        !   JOB ID
     +----------+----------+
 >08 !  JMRNAM  !          !   USER SPECIFIED JOB NAME
     +----------+----------+
     /          /          /
     /          /          /
     +----------+----------+
 >10 !  JMRTID  !  JMRSSZ  !   TASK ID OF INITIAL TASK
     +----------+----------+      SIZE OF JCA  1,2,3
 >12 !  JMRPRM  !          !   TASK BID PARAMETERS
     +----------+----------+
 >14 !          !          !
     +----------+----------+
 >16 !  JMRSID  !  JMRPFL  !   STATION ID OF TASK (JOB)
     +----------+----------+      PROGRAM FILE LUNO OF TASK
 >18 !       JMRSYN        !   SYNONYM SEGMENT SEGMENT ID
     +----------+----------+
 >1A !       JMRLNM        !   LOGICAL NAME BLOCK SEGMENT ID
     +----------+----------+
 >1C !  JMRUID  !          !   USER ID
     +----------+----------+
     /          /          /
     /          /          /
     +----------+----------+
 >24 !  JMRPWD  !          !   PASSWORD
     +----------+----------+
     /          /          /
     /          /          /
     +----------+----------+
 >2C !  JMRACC  !          !   ACCOUNT NUMBER
     +----------+----------+
     /          /          /
     /          /          /
     +----------+----------+
```

FLAGS FOR FIELD: JMRFLG #04 - JOB MANAGER CONTROL FLAGS

```
  JMFNID = (X...............) - NEW USED ID SPECIFIED (CREATE)
  JMFVER = (.X..............) - BYPASS VERFY CHECKS IN JM
  JMFBCH = (..X.............) - BATCH JOB
  JMFRES = (...XXXXXXXX.....) - FLAG BITS 3 - 10 RESERVED
  JMFPVL = (...........XXXXX) - PRIVILEGE LEVEL
```

EQUATES:

LABEL	EQUATE TO	VALUE	DESCRIPTION
JMRSIZ	$	>10	SIZE OF BASIC CALL BLOCK
JMRSZ2	$	>3C	JMR SIZ FOR CREATE OPERATION

```
*************************************************************
*                                                           *
*  JOB STATUS BLOCK            (JSB)        09/09/83        *
*                                                           *
*          LOCATION: SYSTEM AREA                            *
*************************************************************
*  THE JSB PROVIDES THE INFORMATION ABOUT A JOB WHICH IS
*  NEEDED BY DNOS WHETHER OR NOT THE JOB COMMUNICATION AREA
*  IS IN MEMORY.  THIS INFORMATION INCLUDES FLAGS, QUEUE
*  LINKS, STATUS INFORMATION, AND JCA LOCATION DATA.


      *----------+----------*
  >00 !        JSBJSB       !   POINTER TO NEXT JSB
      +----------+----------+
  >02 !        JSBJID       !   JOB ID (UNIQUE TO SITE)
      +----------+----------+
  >04 !  JSBFLG  !  JSBTCT  !   JOB FLAGS
      +----------+----------+      JOB TASK COUNT
  >06 !  JSBPRI  !  JSBSTA  !   JOB PRIORITY
      +----------+----------+      JOB STATE
  >08 !  JSBAPR  !  JSBWPR  !   ACTIVE PRIORITY (HIGHEST)
      +----------+----------+      WAITING ON MEMORY PRI(HIGHEST)
  >0A !        JSBQL        !   ACTIVE QUEUE LINK
      +----------+----------+
  >0C !        JSBWOM       !   LINK FOR WAITING MEMORY QUEUE
      +----------+----------+
  >0E !        JSBEOR       !   END OF REQUEST PROCESSING ANCHOR
      +----------+----------+
  >10 !        JSBJCA       !   SSB ADDRESS FOR JCA
      +----------+----------+
  >12 !        JSBSMT       !   SM TABLE SSB ADDRESS FOR JCA
      +----------+----------+
  >14 !  JSBNAM  !          !   JOB NAME
      +----------+----------+
      /          /          /
      /          /          /
      +----------+----------+
  >1C !  JSBUID  !          !   USER ID OF JOB
      +----------+----------+
      /          /          /
      /          /          /
      +----------+----------+
  >24 !        JSBWOT       !   TABLE AREA JSB WAIT QUEUE LINK
      +----------+----------+
  >26 !        JSBVER       !   PTR TO SELF FOR VERIFICATION
      +----------+----------+


  FLAGS FOR FIELD: JSBFLG     #04 - JOB FLAGS

    JSFVER = (X...............) - BY-PASS VERIFICATION CHECKS
    JSFACC = (.X..............) - ACCOUNTING STARTED FOR JOB
    JSFBAC = (..X.............) - BACKGROUND JOB
```

EQUATES:

LABEL	EQUATE TO	VALUE	DESCRIPTION
JSBSIZ	$	>28	

```
************************************************************
*                                                          *
*  KIF CURRENCY BLOCK    (KCB)                  01/22/82   *
*                                                          *
*          LOCATION: SYSTEM TABLE AREA                     *
*                                                          *
************************************************************
* THE KCB IS USED TO MAINTAIN CURRENCY INFORMATION ABOUT A
* KEY INDEXED FILE IN USE.  THE KCB IS BUFFERED ALONG WITH
* THE IRB DESCRIBING THE I/O REQUEST.
*
*  SPECIAL FIELD COMMENTS:
*  KCBKAD - FIRST TWO WORDS GIVE THE PHYSICAL RECORD NUMBER
*           OF THE LOGICAL RECORD.  THE THIRD WORD IS THE
*           ID OF THE LOGICAL RECORD.
*  KCBBTP - FIRST TWO WORDS GIVE THE PHYSICAL RECORD NUMBER
*           OF THE KEY FROM WHICH THE CURRENCY WAS CREATED.
*           THE THIRD WORD IS THE LOGICAL ADDRESS OF THE KEY
*           WHEN THE PHYSICAL RECORD IS MAPPED INTO KIF
*           PROCESSING CODE.


      *----------+----------*
  >02 !  KCBINF  !  KCBKNM  !   CURRENCY INFORMATION CODE
      +----------+----------+      KEY NUMBER
  >04 !        KCBKAD       !   KEY ADDRESS
      +----------+----------+
  >06 !        KCBDBK       !   DATA BASE KEY (3 WORDS)
      +----------+----------+
  >08 !        FILL00       !
      +----------+----------+
  >0A !        FILL01       !
      +----------+----------+
  >0C !        KCBBTP       !   B-TREE POINTER (3 WORDS)
      +----------+----------+
  >0E !        FILL02       !
      +----------+----------+
  >10 !        FILL03       !
      +----------+----------+
  >12 !        KCBBES       !   B-TREE ENTRY SIZE
      +----------+----------+
  >14 !  KCBLOC  !  KCBCOC  !   LAST OPCODE USED
      +----------+----------+      CURRENT OPCODE


  EQUATES:

     LABEL   EQUATE TO   VALUE   DESCRIPTION
     -----   ---------   -----   ------------------------------
     KCBSIZ  $-KCBINF      >14
```

```
************************************************************
*                                                          *
*  KEY DESCRIPTOR BLOCK  (KDB)                  09/10/79   *
*                                                          *
*           LOCATION: STA (PART OF IRB)                    *
************************************************************
* THE KDB IS PART OF A CREATE KEY INDEXED FILE I/O REQUEST
* BRB WHICH DESCRIBES THE KEYS TO BE CREATED.


      *----------+----------*
  >00 !        KDBOVH       !   OVERHEAD
      +----------+----------+
  >02 !        KDBMLR       !   MAX NUMBER LOGICAL RECORDS
      +----------+----------+
  >04 !                     !
      +----------+----------+
  >06 !        KDBNKY       !   NUMBER OF KEYS
      +----------+----------+
  >08 !  KDBOFF  !          !   SPACE FOR MAXIMUM # KEYS
      +----------+----------+
      /          /          /
      /          /          /
      +----------+----------+

                                DESCRIPTION OF ONE KEY
      *----------+----------*
  >00 !  KDBFGS  !  KDBSIZ  !   FLAGS
      +----------+----------+      NUMBER OF CHARACTERS IN KEY
  >02 !        KDBO         !   KEY OFFSET IN RECORD
      +----------+----------+


  FLAGS FOR FIELD: KDBFGS     #00 - FLAGS

          = (XXX.............) - *** RESERVED ***
    KDBPLG = (...X............) - BIT 3 SET IF PARTIAL LOGGING
    KDB33  = (....X...........) - BIT 4 SET IF SEQUENTIAL KIF
    KDBOFG = (.....X..........) - BIT 5 SET IF KEY IS OPTIONAL
    KDBSFG = (......X.........) - BIT 6 SET IF SEQUENTIAL CMNDS
    KDBDFG = (.......X........) - BIT 7 SET IF DUPLICATES OK


  EQUATES:

     LABEL   EQUATE TO   VALUE   DESCRIPTION
     -----   ---------   -----   ------------------------------
     KDBMKY  14            >0E     MAXIMUM # OF KEYS IN FILE
     KDBMSZ  100           >64     MAXIMUM KEY SIZE
     KDBNXT  $             >04     SIZE OF KEY DESCRIPTOR
```

```
**********************************************************
*                                                        *
* KEY INDEXED FILE KEY DESCRIPTOR RECORD(KDR)  09/09/83  *
*                                                        *
*           LOCATION: DISK RESIDENT STRUCTURE            *
**********************************************************
* THE KDR DESCRIBES THE KEYS OF A KEY INDEXED FILE.  THE
* FIELD AT KDROFF IS ONE OR MORE REPLICATIONS OF THE
* FIELDS BEGINNING AT KDRFGS.


      *----------+----------*
  >00 !        KDRHKC       !   HASH KEY COUNT
      +----------+----------+
  >02 !        KDRHKV       !   HASH KEY VALUE
      +----------+----------+
  >04 !        FILL00       !   (WORD COPIED FROM KDB)
      +----------+----------+
  >06 !        KDRNKY       !   NUMBER OF KEYS
      +----------+----------+
  >08 !  KDROFF  !          !   SPACE FOR MAXIMUM # KEYS
      +----------+----------+
      /          /          /
      /          /          /
      +----------+----------+
  >40 !  KDRCD   !          !   CREATION DATE AND TIME USED
      +----------+----------+
  >42 !          !          !
      +----------+----------+
  >44 !          !          !
      +----------+----------+
  >46 !  KDRSEQ  !  KDRCCT  !   CONCATENATED SET SEQUENCE NUM.
      +----------+----------+      TOTAL CONCAT. FILES IN SET

                              FLAGS DESCRIPTION (NOT A VARIANT)
      *----------+----------*
  >00 !  KDRFGS  !  KDRSIZ  !   FLAGS
      +----------+----------+      # CHARS IN KEY
  >02 !        KDRO         !   KEY OFFSET IN RECORD
      +----------+----------+


  FLAGS FOR FIELD: KDRFGS    #00 - FLAGS

           = (XXX.............) - *** RESERVED ***
    KDRPFG = (...X............) - BIT 3 SET IF PARTIAL LOGGING
    KDR33  = (....X...........) - BIT 4 SET IF SEQUENTIAL KIF
    KDROFG = (.....X..........) - BIT 5 SET IF KEY IS OPTIONAL
    KDRSFG = (......X.........) - BIT 6 SET IF SEQUENTIAL CMNDS
    KDRDFG = (.......X........) - BIT 7 SET IF DUPLICATES OK
```

EQUATES:

LABEL	EQUATE TO	VALUE	DESCRIPTION
KDRMKY	14	>0E	MAX # OF KEYS IN FILE
KDRMSZ	100	>64	MAX KEY SIZE
KDRNXT	$	>04	SIZE OF KEY DESCRIPTOR

```
************************************************************
*                                                          *
*  KIF INFORMATION BLOCK     (KIB)              02/26/80   *
*                                                          *
*            LOCATION: DISK AND BUFFER SEGMENT             *
************************************************************
* THE KIB DESCRIBES A KEY INDEXED FILE DATA BLOCK.
*
* SPECIAL FIELD COMMENTS:
* KIBBLK -  THE PHYSICAL RECORD NUMBER OF THIS BLOCK.  THIS
*           FIELD IS MAINTAINED SO THAT IF A SYSTEM CRASH
*           OCCURS WHILE THIS BLOCK IS BEING MODIFIED, THE
*           LOGGED IMAGE CAN BE RESTORED TO THE CORRECT FILE
*           RECORD.
* KIBCMD -  THE OPCODE OF THE CURRENT COMMAND.  THIS IS
*           MAINTAINED FOR LOGGING PURPOSES.
* KIBSR  -  THE NUMBER OF BYTES REMAINING IN THE PHYSICAL
*           RECORD.
* KIBFCB -  THIS FIELD IS USED TO LINK THE BLOCK ON THE FREE
*           BLOCK CHAIN.
* KIBRSZ -  THE SIZE IN BYTES OF THE FIRST LOGICAL RECORD
*           INCLUDING THIS WORD.


      *----------+----------*
  >00 !       KIBBLK        !   BLOCK NUMBER
      +----------+----------+
  >02 !                     !
      +----------+----------+
  >04 !       KIBCMD        !   COMMAND NUMBER
      +----------+----------+
  >06 !       KIBSR         !   SPACE REMAINING IN BYTES
      +----------+----------+
  >08 !       KIBFCB        !   FREE CHAIN POINTER
      +----------+----------+
  >0A !                     !
      +----------+----------+
  >0C !       KIBHID        !   HIGHEST LOGICAL RECORD ID USED
      +----------+----------+
  >0E !       KIBRSZ        !   RECORD SIZE OF 1ST RECORD
      +----------+----------+
  >10 !       KIBRID        !   ID OF FIRST LOGICAL RECORD
      +----------+----------+
```

EQUATES:

LABEL	EQUATE TO	VALUE	DESCRIPTION
KIBOFB	$	>08	OVERFLOW BLOCK POINTER

```
**********************************************************
*                                                        *
*  KIF TASK AREA         (KIT)                 01/21/82  *
*                                                        *
*           LOCATION: SYSTEM TABLE AREA                  *
*           (USED ONLY BY ASSEMBLY LANGUAGE CODE)        *
**********************************************************
* THE KIT IS ATTACHED TO THE FILE MANAGEMENT WORK AREA (FWA)
* FOR ADDITIONAL WORKING STORAGE FOR KIF PROCESSING.  IT
* INCLUDES INFORMATION ABOUT THE CURRENT REQUEST, THE STATE
* OF THE FILE, AND SEVERAL FIELDS OF THE FCB TO MINIMIZE
* MAPPING DURING PROCESSING.
**********************************************************
*                                                        *
*  FILE MANAGER WORK AREA     (FWA)            01/21/82  *
*                                                        *
*           LOCATION: SYSTEM AREA                        *
**********************************************************
*  THE FWA IS USED BY FILE MANAGEMENT AND BY KIF MANAGEMENT
*  AS A GENERAL WORK AREA.  R15 POINTS TO THE FWA.


      *----------+----------*
  >00 !  FWAWP   !          !   WORKSPACE USED BY FM
      +----------+----------+
      /          /          /
      /          /          /
      +----------+----------+
  >20 !        FWAFLG       !   MIDDLE SEGMENT FLAGS
      +----------+----------+
  >22 !        FWATCT       !   MULTIRECORD CHARS TRANSFERRED
      +----------+----------+
  >24 !        FWAOAD       !   CURRENT OVERLAY AREA ADDRESS
      +----------+----------+
  >26 !        FWAPC        !   SAVED PROGRAM COUNTER
      +----------+----------+
  >28 !        FWAXWP       !   BLWP VECTOR FOR RETURNING
      +----------+----------+
  >2A !        FWAXPC       !
      +----------+----------+
  >2C !        FWABN        !   SAVED RPBBN (2 WORDS)
      +----------+----------+
  >2E !                     !
      +----------+----------+
  >30 !        FWAOCB       !   SAVED RPBOCB
      +----------+----------+
  >32 !        FWALFG       !   SAVED LDT FLAGS
      +----------+----------+
  >34 !        FWAFFC       !   FIRST FCB FOR CC FILES
      +----------+----------+
  >36 !        FWAFMT       !   SSB FOR THE FMT WITH THE FCB
      +----------+----------+
  >38 !       FWAFCB         !    FCB ADDRESS IN THE FMT
      +----------+----------+
  >3A !       FWABST         !    SMT SSB ADDR FOR BUFFER
      +----------+----------+
  >3C !       FWABSB         !    SSB ADDR FOR BUFFER
      +----------+----------+
  >3E !       FWAPRS         !    PHYSICAL RECORD SIZE
      +----------+----------+
  >40 !       FWAUBT         !    USER BUFFER SMT SSB ADDR
      +----------+----------+
  >42 !       FWAUBS         !    USER BUFFER SSB ADDRESS
      +----------+----------+
  >44 !       FWAUBO         !    USER BUFFER OFFSET
      +----------+----------+
  >46 !       FWAUBL         !    USER BUFFER LENGTH
      +----------+----------+
  >48 !       FWAFMB         !    FMT BIAS
      +----------+----------+
  >4A !       FWARN1         !    RECORD # RELATIVE TO CURRENT
      +----------+----------+
  >4C !       FWARN2         !    FILE OF CONCATENATED SET
      +----------+----------+
  >4E !       FWAOOB         !    OLD OFFSET IN USER BUFFER
      +----------+----------+
  >50 !       FWAUBR         !    BUFFER LENGTH REMAINING
      +----------+----------+
  >52 !       FWAFFG         !    FILE MGR FLAGS
      +----------+----------+
  >54 !  FWASTK  !          !    STACK AREA
      +----------+----------+
      /          /          /
      /          /          /
      +----------+----------+

                                TEMPORARY KIF STORAGE IN TASK AREA
      *----------+----------*
  >54 !  FILL01  !          !    KIF STACK
      +----------+----------+
      /          /          /
      /          /          /
      +----------+----------+
>0130 !       KITBKM         !    LOGICAL BLOCK END OF MEDIUM
      +----------+----------+
>0132 !                      !
      +----------+----------+
>0134 !       KITCMD         !    CURRENT COMMAND NUMBER
      +----------+----------+
>0136 !       KITEBQ         !    EMPTY BLOCK QUEUE HEAD
      +----------+----------+
>0138 !                      !
      +----------+----------+
>013A !       KITCLB         !    CURRENT LOG BLOCK
      +----------+----------+
>013C !        KITFBQ         !   FREE BLOCK QUEUE HEAD
      +-----------+-----------+
>013E !                       !
      +-----------+-----------+
>0140 !        KITBTR         !   B-TREE ROOTS
      +-----------+-----------+
>0142 !  KITKDB   !           !   KDB OF CURRENT REQUEST
      +-----------+-----------+
      /           /           /
      /           /           /
      +-----------+-----------+
>017C !        KBUF1A         !   ADDRESS OF FIRST KEY BUFFER
      +-----------+-----------+
>017E !        KBUF2A         !   ADDRESS OF SECOND KEY BUFFER
      +-----------+-----------+
>0180 !        KBUF3A         !   ADDRESS OF THIRD KEY BUFFER
      +-----------+-----------+
>0182 !        KEYNUM         !   KEY # OF KEY CURRENTLY USING
      +-----------+-----------+
>0184 !        KEYSZ          !   SIZE (CHARS) OF THIS KEY
      +-----------+-----------+
>0186 !  BTSTK    !           !   B-TREE STACK
      +-----------+-----------+
      /           /           /
      /           /           /
      +-----------+-----------+
>01C2 !        BTSTKA         !   ADDR 1ST ENTRY OF B-TREE STACK
      +-----------+-----------+
>01C4 !        BTSPTR         !   ADDR 1ST UNUSED B-T STACK ENTRY
      +-----------+-----------+
>01C6 !        NEIBTS         !   NUMBER ENTRIES IN B-TREE STACK
      +-----------+-----------+
>01C8 !        TS1L2          !   1 WORD OF LEVEL 1 TEMP STORAGE
      +-----------+-----------+
>01CA !        T1L2A          !   ADDRESS OF TS1L4
      +-----------+-----------+
>01CC !        TS1L2A         !   1 WORD OF LEVEL 1 TEMP STORAGE
      +-----------+-----------+
>01CE !        TS1L2B         !   1 WORD OF LEVEL 1 TEMP STORAGE
      +-----------+-----------+
>01D0 !        TS1L2C         !   1 WORD OF LEVEL 1 TEMP STORAGE
      +-----------+-----------+
>01D2 !        T1L2CA         !   ADDRESS OF TS1L2C
      +-----------+-----------+
>01D4 !        TS1L2D         !   1 WORD OF LEVEL 1 TEMP STORAGE
      +-----------+-----------+
>01D6 !        T1L2DA         !   ADDRESS OF TS1L2D
      +-----------+-----------+
>01D8 !        TS1L4          !   2 WORDS OF LEVEL 1 TEMP STORAGE
      +-----------+-----------+
>01DA !                       !
      +-----------+-----------+
>01DC !        T1L4A          !   ADDRESS OF TSILY
```

```
      +----------+----------+
>01DE !        TS1L4A       !   2 WORDS OF LEVEL 1 TEMP STORAGE
      +----------+----------+
>01E0 !                     !
      +----------+----------+
>01E2 !        T1L4AA       !   ADDRESS OF TS1L4A
      +----------+----------+
>01E4 !        TS1L4B       !   2 WORDS OF LEVEL 1 TEMP STORAGE
      +----------+----------+
>01E6 !                     !
      +----------+----------+
>01E8 !        T1L4BA       !   ADDRESS OF TS1L4B
      +----------+----------+
>01EA !        TS1L6        !   3 WORDS OF LEVEL 1 TEMP STORAGE
      +----------+----------+
>01EC !                     !
      +----------+----------+
>01EE !                     !
      +----------+----------+
>01F0 !        T1L6A        !   ADDRESS OF TS1L6
      +----------+----------+
>01F2 !        TS2L2        !   1 WORD OF LEVEL 2 TEMP STORAGE
      +----------+----------+
>01F4 !        T2L2A        !   ADDRESS OF TS2L2
      +----------+----------+
>01F6 !        TS2L4        !   2 WORDS OF LEVEL 2 TEMP STORAGE
      +----------+----------+
>01F8 !                     !
      +----------+----------+
>01FA !        T2L4A        !   ADDRESS OF TS2L4
      +----------+----------+
>01FC !        TS2L4A       !   2 WORDS OF LEVEL 2 TEMP STORAGE
      +----------+----------+
>01FE !                     !
      +----------+----------+
>0200 !        T2L4AA       !   ADDRESS OF TS2L4A
      +----------+----------+
>0202 !        TS2L4B       !   2 WORDS OF LEVEL 2 TEMP STORAGE
      +----------+----------+
>0204 !                     !
      +----------+----------+
>0206 !        T2L4BA       !   ADDRESS OF TS2L4B
      +----------+----------+
>0208 !        TS2L4C       !   2 WORDS OF LEVEL 2 TEMP STORAGE
      +----------+----------+
>020A !                     !
      +----------+----------+
>020C !        T2L4CA       !   ADDRESS OF TS2L4C
      +----------+----------+
>020E !        RQDBKA       !   ADDRESS OF KCBDBK
      +----------+----------+
>0210 !        RQBTPA       !   ADDRESS OF KCBBTP
      +----------+----------+
>0212 !       BTDBKA        !    ADDRESS OF BTBDBK
      +----------+----------+
>0214 !       FDRCFG        !    FDR CHANGE FLAG
      +----------+----------+
>0216 !       BTSPLT        !    B-TREE SPLIT FLAG
      +----------+----------+
>0218 ! TBTP     !          !    TEMPORARY B-TREE POINTER
      +----------+----------+
>021A !          !          !
      +----------+----------+
>021C !          !          !
      +----------+----------+
>021E !       ATBTP         !    ADR TEMPORARY B-TREE POINTER
      +----------+----------+
>0220 ! TDBK     !          !    TEMPORARY DATA BASE KEY
      +----------+----------+
>0222 !          !          !
      +----------+----------+
>0224 !          !          !
      +----------+----------+
>0226 !       ATDBK         !    ADR OF TEMPORARY DATA BASE KEY
      +----------+----------+
>0228 !       SBLK          !    SAVED SUCCESSOR BLOCK NUMBER
      +----------+----------+
>022A !                     !
      +----------+----------+
>022C ! ASBLK    !          !    ADR SAVED SUCCESSOR BLK NUMBER
      +----------+----------+
>022E ! LEAFFL   !          !    SAVED LEAF FLAG
      +----------+----------+
>0230 !       RWSAV8        !    SAVE R8 HERE
      +----------+----------+
>0232 !       RRPDC         !    DUP COUNT OF DUP WANTED
      +----------+----------+
>0234 !       RRPDFL        !    DUPLICATES FLAG
      +----------+----------+
>0236 !       RQSAVE        !    ORIG RQ FOR PASSIVE READS
      +----------+----------+
>0238 ! CWSR0    !  FILL02  !    ALT SEQ=>FF, STD SEQ=>00
      +----------+----------+       CONV REQ FLAG (BIT 8)
>023A !       CWSR1         !    STORAGE FOR KEY1 ADDRESS
      +----------+----------+
>023C !       CWSR2         !    STORAGE FOR KEY2 ADDRESS
      +----------+----------+
>023E !       CWSR3         !    STORAGE FOR KEY LENGTH
      +----------+----------+
>0240 !       CWSR4         !    KMCNV BASE DATA ADDRESS
      +----------+----------+
>0242 !       CWSR5         !    KMUCV BASE DATA ADDRESS
      +----------+----------+
```

FLAGS FOR FIELD: FWAFFG   #52 - FILE MGR FLAGS

```
  FWAPOP = (X...............) - PASSIVE OPERATION FLAG
  FWAQW  = (.X..............). - QUEUED TO WAITING QUEUE IN FCB
```

EQUATES:

LABEL	EQUATE TO	VALUE	DESCRIPTION
FWASIZ	$	>F4	SIZE OF FWA INCLUDING WSP
MAXSSZ	9	>09	ONLY 9 STACK ENTRIES ALLOWED
KITSIZ	$	>244	SIZE INCLUDING WORKSPACE

```
************************************************************
*                                                          *
*  KEYBOARD STATUS BLOCK      (KSB)              09/28/79  *
*                                                          *
*            LOCATION: SYSTEM AREA                         *
************************************************************
* THE KSB IS APPENDED TO A PHYSICAL DEVICE TABLE (PDT) FOR
* A KEYBOARD DEVICE.  IT IS USED BY THE DEVICE SERVICE
* ROUTINE (DSR) AS A WORKSPACE WHILE HANDLING THE KEYBOARD.

                                CHARACTER BUFFER LENGTH
      *----------+----------*
  >00 !        KSBPDT        !   R0  - PDT POINTER
      +----------+----------+
  >02 !        KSBQOC        !   R1  - QUEUE OUTPUT COUNT
      +----------+----------+
  >04 !        KSBQIP        !   R2  - QUEUE INPUT POINTER
      +----------+----------+
  >06 !        KSBQOP        !   R3  - QUEUE OUTPUT POINTER
      +----------+----------+
  >08 !        KSBQEP        !   R4  - QUEUE END POINTER
      +----------+----------+
  >0A !        KSBCRQ        !   R5  - GET CHAR REQUEST QUEUE
      +----------+----------+
  >0C !  KSBFL   !  KSBSN    !   R6  - KSB FLAGS
      +----------+----------+       - STATION NUMBER
  >0E !        KSBR7         !   R7  - SCRATCH
      +----------+----------+
  >10 !        KSBTSB        !   R8  - TSB ADDRESS OF CHAR OWNER
      +----------+----------+
  >12 !        KSBR9         !   R9  - SCRATCH
      +----------+----------+
  >14 !        KSBR10        !   R10 - SCRATCH
      +----------+----------+
  >16 !        KSBR11        !   R11 - SCRATCH
      +----------+----------+
  >18 !        KSBCRU        !   R12 - CRU BASE
      +----------+----------+
  >1A !        KSBR13        !   R13 - SAVED WORKSPACE POINTER
      +----------+----------+
  >1C !        KSBR14        !   R14 - SAVED PROGRAM COUNTER
      +----------+----------+
  >1E !        KSBR15        !   R15 - SAVED STATUS
      +----------+----------+


  FLAGS FOR FIELD: KSBFL      #0C - R6  - KSB FLAGS

    KSBCHM = (X...............) - CHARACTER MODE
    KSBCIE = (.X..............) - SCI ENABLED
    KSBRCM = (..X.............) - RECORD MODE
    KSBCIB = (...X............) - SCI BID IN PROCESS
  KSBICP = (....X...........) - SCI ACTIVE
  KSBSET = (.....X..........) - COMMAND I/O HOLD
  KSBKIO = (......X.........) - COMMAND I/O ABORT
  KSBBRK = (.......X........) - DEACTIVATE BREAK KEY
```

EQUATES:

LABEL	EQUATE TO	VALUE	DESCRIPTION
KSBCBL	6	>06	CHARACTER BUFFER LENGTH
KSBSIZ	$	>20	

```
************************************************************
*                                                          *
*  LOGICAL DEVICE TABLE       (LDT)               01/27/83 *
*                                                          *
*           LOCATION: SYSTEM AREA AND JCA                  *
************************************************************
* THE LDT CONTAINS INFORMATION DESCRIBING AN I/O RESOURCE TO
* WHICH A LOGICAL UNIT NUMBER (LUNO) HAS BEEN ASSIGNED.  IT
* INCLUDES TYPE FLAGS, OWNERSHIP, AND STATE INFORMATION.
                        ** BEGINNING PACKED RECORD LDT


      *----------+----------*
  >00 !        LDTLDT       !   LINK TO NEXT LDT
      +----------+----------+
  >02 !  LDTIOC  !  LDTLUN  !   INITIATE I/O COUNT
      +----------+----------+      LOGICAL UNIT NUMBER


      *----------+----------*
  >04 !  LDTTYP  !  LDTTF   !   I/O RESOURCE TYPE
      +----------+----------+      FLAGS FOR LDT TYPE (SEE LDTXFL)
  >06 !        LDTFLG       !   FLAGS
      +----------+----------+
  >08 !        LDTRLK       !   RESOURCE LINK: FCA, PDT OR CCB
      +----------+----------+
  >0A !        LDTTSB       !   OWNER TSB LIST ANCHOR
      +----------+----------+
  >0C !        LDTJSB       !   OWNER JSB ADDRESS
      +----------+----------+

                           DEVICE, CHANNEL LDT
      *----------+----------*
  >0E !        LDTSID       !   SESSION ID
      +----------+----------+

                           FILE LDT
      *----------+----------*
  >0E !        LDTFMT       !   SSB FOR THE FMT
      +----------+----------+
  >10 !        LDTFCB       !   FCB ADDRESS IN THE FMT
      +----------+----------+
  >12 !        LDTCAR       !   COMPOSITE ACCESS RIGHTS
      +----------+----------+


      *----------+----------*
  >04 !        LDTXFL       !   RESOURCE TYPE / FLAGS
      +----------+----------+
  >14 SIZE              ** END OF PACKED RECORD
```

```
   FLAGS FOR FIELD: LDTFLG     #06 - FLAGS

      LDFDEL = (X...............) - LDT IS DELETE PROTECTED
      LDFFWT = (.X..............) - FORCED (IMMEDIATE) WRITE BIT
      LDFCBA = (..X.............) - CREATED BY ASSIGN BIT
      LDFNUS = (...X............) - LDT IS CURRENTLY NON-USABLE
      LDFPRM = (....X...........) - PARAMETERS ARE PRESENT
      LDFUBI = (.....X..........) - UNBLOCKED(1)/BLOCKED(0) OPEN
      LDFACU = (......XX........) - ACCESS PRIV. IN USE
*                                   00 = EXCLUSIVE WRITE
*                                   01 = EXCLUSIVE ALL
*                                   10 = SHARED
*                                   11 = READ ONLY
      LDFSC1 = (........XX......) - LDT SCOPE (TASK, JOB, GLOBAL)
*                                   00 = TASK-LOCAL
*                                   01 = JOB-LOCAL
*                                   10 = GLOBAL
*                                   11 = SHARED
      LDFDWE = (..........X.....) - DEFERRED WRITE ERROR
      LDFVNT = (...........X....) - EVENTS REQUESTED (KB DEVICES)
      LDFUSD = (............X...) - LDT HAS BEEN USED
      LDFDIA = (.............X..) - DIAGNOSTIC STATE
             = (..............XX) - *** RESERVED ***


   FLAGS FOR FIELD: LDTCAR     #12 - COMPOSITE ACCESS RIGHTS

      LDFRDF = (X...............) - READ ACCESS FLAG
      LDFWRF = (.X..............) - WRITE ACCESS FLAG
      LDFDLF = (..X.............) - DELETE ACCESS FLAG
      LDFEXF = (...X............) - EXECUTE ACCESS FLAG
      LDFCTF = (....X...........) - CONTROL ACCESS FLAG
             = (.....XXXXXXXXXXX) - **RESERVED**


   FLAGS FOR FIELD: LDTXFL     #04 - RESOURCE TYPE / FLAGS

             = (XXXXXXXX........) - ** ACTUALLY LDTTYP FIELD **
      LDFJLO = (........X.......) - 1=JOB LEVEL OPEN
             = (.........XX.....) - **RESERVED**
      LDFVD  = (...........X....) - LDT FOR A VIRTUAL DEVICE
      LDFREM = (............X...) - LDT FOR REMOTE RESOURCE
      LDFCHN = (.............X..) - LDT FOR CHANNEL
      LDFDEV = (..............X.) - LDT FOR DEVICE
      LDFFIL = (...............X) - LDT FOR FILE
*------
* VALUES FOR I/O RESOURCE TYPE (FIELD LDTTYP)
*------
*

   EQUATES:
```

LABEL	EQUATE TO	VALUE	DESCRIPTION
LDFACM	>0300	>300	ACCESS PRIV. BIT MASK
LDFSCM	>00C0	>C0	SCOPE BIT MASK
LDFMM	LDFDWE	>0A	MAGIC MODE FOR EVDT
LDTSZ1	$	>0E	
LDTSZ2	$	>10	DEVICE, CHANNEL LDT SIZE
LDTFSZ	$	>14	FILE LDT SIZE
LDTRES	0	>00	RESERVED
LDTSEQ	1	>01	SEQUENTIAL
LDTRR	2	>02	RELATIVE RECORD
LDTKIF	3	>03	KEY INDEX
LDTDIR	4	>04	DIRECTORY
LDTPRG	5	>05	PROGRAM
LDTIMG	6	>06	IMAGE
LDTDMY	0	>00	DUMY
LDTSD	1	>01	SPECIAL DEVICES
LDTKSR	2	>02	KSR
LDTASR	3	>03	ASR
LDTCS	4	>04	CASSETTE
LDTRS2	5	>05	RESERVED
LDTDK	6	>06	SINGLE DENSITY DISKETTE
LDTDS	7	>07	DISK
LDTMT	8	>08	MAG TAPE (979,CARTRIDGE)
LDTTPD	9	>09	820
LDTV11	>A	>0A	911 VDT
LDTLPS	>B	>0B	LINE PRINTER (SERIAL)
LDTLPP	>C	>0C	LINE PRINTER (PARALLEL)
LDTC3Q	>D	>0D	COMM 9903 (FCCC)
LDTCOM	>E	>0E	COMM I/F
LDTIND	>F	>0F	INDUSTRIAL DEVICES
LDTCR	>10	>10	CARD READER
LDT940	>11	>11	940 EVT
LDT931	>12	>12	931 EVT
LDTEN	>13	>13	ETHERNET
LDTBCM	>14	>14	BCAIM
LDTVT	>15	>15	VIRTUAL TERMINAL BASE

```
************************************************************
*                                                          *
*  LOG FILE DEFINITION       (LFD)             04/26/82    *
*                                                          *
*            LOCATION:  SYSTEM ROOT                        *
************************************************************
* THE LFD IS BUILT DURING SYSTEM GENERATION AND IS USED TO
* KEEP TRACK OF THE STATE OF THE SYSTEM LOG OPTIONS.


      *----------+----------*
  >00 !  LFDFLG  !  LFDERR  !   FLAGS
      +----------+----------+      ERROR BYTE FOR RECREATE TASK
  >02 !       LFDMAX        !   MAX MESSAGE COUNT (0 = NONE)
      +----------+----------+
  >04 !  LFDTID  !  LFDTDU  !   TASK ID TO BID FOR FULL FILES
      +----------+----------+      USER TASK ID TO BID ON FULL
  >06 !  LFDDNM  !          !   LOG DEVICE NAME  (' '= NONE)
      +----------+----------+
  >08 !          !          !
      +----------+----------+
  >0A !  LFDFN1  !          !   FILENAME 1
      +----------+----------+
      /          /          /
      /          /          /
      +----------+----------+
  >12 !  LFDFN2  !          !   FILENAME 2
      +----------+----------+
      /          /          /
      /          /          /
      +----------+----------+
  >1A !       LFDALC        !   LOG FILE ALLOCATION
      +----------+----------+
  >1C !  LFDLUN  !          !   LUNOS
      +----------+----------+


  FLAGS FOR FIELD: LFDFLG    #00 - FLAGS

    LDFFDS = (X...............) - FILES DISABLED
    LDFDDS = (.X..............) - DEVICE DISABLED
    LDF2ND = (..X.............) - CURRENTLY USING 2ND FILE
    LDFCSH = (...X............) - CRASH FILE PROCESSED
    LDFRCR = (....X...........) - RECREATE FILES
    LDFIMW = (.....X..........) - FILES ARE IMMEDIATE WRITE
    LDFSBE = (......X.........) - SUPPRESS BID ERROR LOGGING
```

EQUATES:

LABEL	EQUATE TO	VALUE	DESCRIPTION
LFDSIZ	$	>1E	

```
************************************************************
*                                                          *
*  LINE PRINTER PDT EXTENSION (LPD)               04/21/82 *
*                                                          *
*             LOCATION: SYSTEM AREA                        *
************************************************************
* THE LPD IS AN EXTENSION TO THE PHYSICAL DEVICE TABLE (PDT)
* FOR A LINE PRINTER.  IT CONTAINS POINTERS AND FLAGS USED
* BY THE LINE PRINTER DEVICE SERVICE ROUTINE (DSR).


      *----------+----------*
  >00 !        LPDIFF       !   INTERFACE FLAGS
      +----------+----------+
  >02 !        LPDQCC       !   QUEUE CHARACTER COUNT
      +----------+----------+
  >04 !        LPDQIP       !   QUEUE INPUT POINTER
      +----------+----------+
  >06 !        LPDQOP       !   QUEUE OUTPUT POINTER
      +----------+----------+
  >08 !        LPDQEP       !   QUEUE END POINTER
      +----------+----------+
  >0A !  LPDBUF  !          !   CHARACTER BUFFER
      +----------+----------+
  >0C !          !          !
      +----------+----------+
  >0E !        LPDQSZ       !   CHARACTER QUEUE SIZE
      +----------+----------+
  >10 !  LPDSPX  !  LPDSPR  !   TRANSMIT SPEED
      +----------+----------+      RECEIVE SPEED

  FLAGS FOR FIELD: LPDIFF     #00 - INTERFACE FLAGS

    LPFIF  = (X...............) - INTERFACE  (0<=DM; 0>EIA)
    LPFUC  = (.X..............) - UPPERCASE ONLY (0=YES; 1=NO)
    LPFBSY = (..X.............) - "RO" TERMINAL BUSY(0=NO ,1=YES)
    LPF902 = (...X............) - 9902 INTERFACE FLAG
    LPFEOR = (....X...........) - END-OF-RECORD FLAG
*                     1 = SAFE TO ISSUE ENDRCD
*                     0 = DON'T ISSUE ENDRCD
           = (.....XXXXXXXXXXX) - RESERVED


  EQUATES:

     LABEL   EQUATE TO   VALUE   DESCRIPTION
     -----   ---------   -----   ------------------------------
     LPDSIZ   $            >12
```

```
************************************************************
*                                                          *
*  LOAD SEGMENT ENTRY          (LSE)             04/04/79  *
*                                                          *
*            LOCATION: JCA                                 *
************************************************************
* THE LSE DESCRIBES A SEGMENT WHICH IS LOADED INTO MEMORY
* WHILE THIS TASK IS RUNNING, BUT MAY NOT CURRENTLY BE MAPPED
* IN TO THE TASK.  IT IS LINKED TO THE TSB.


      *----------+----------*
  >00 !        LSELSE       !   LINK TO NEXT LOAD SEGMENT ENTRY
      +----------+----------+
  >02 !        LSESSB       !   SSB ADDRESS OF LOADED SEGMENT
      +----------+----------+
  >04 !        LSESMT       !   SM TABLE AREA SSB ADDR.
      +----------+----------+

  EQUATES:

     LABEL   EQUATE TO   VALUE   DESCRIPTION
     -----   ---------   -----   ------------------------------
     LSESIZ  $             >06
```

```
***********************************************************
*                                                         *
*  MASTER READ/MASTER WRITE BUFFER    (MRB)      04/04/83 *
*                                                         *
*            LOCATION: TASK AREA                          *
*                                                         *
***********************************************************
* THE MRB IS A DESCRIPTION OF THE DATA BUFFER RETURNED TO A
* CHANNEL OWNER TASK IN ITS MASTER READ BUFFER.  THIS SAME
* BUFFER STRUCTURE IS USED IN THE MASTER WRITE OPERATION OF
* THE OWNER TASK.  ALL BUFFER POINTERS IN THE MRB ARE
* RELATIVE OFFSETS FROM THE BEGINNING OF THE MRB, RATHER
* THAN BEING ABSOLUTE ADDRESSES.
* MRB VARIANTS ARE PROVIDED FOR THE MAJOR TYPES OF I/O CALL
* BLOCKS: BASIC FILE I/O, I/O WITH REPLY INFORMATION, I/O
* WITH VALIDATION, VDT EXTENSIONS, AND UTILITY OPERATIONS.
* THERE IS ALSO A VARIANT FOR ABORT I/O CALLS.
*
                          ** BEGINNING PACKED RECORD MRB


      *----------+----------*
  >00 !        MRBSID       !    SECURITY INFORMATION (SESSION ID)
      +----------+----------+
  >02 !        MRBRCB       !    REQUESTOR CALL BLOCK ADDRESS
      +----------+----------+
  >04 !        MRBTSB       !    REQUESTOR TSB ADDRESS
      +----------+----------+
  >06 !        MRBJSB       !    REQUESTOR JSB ADDRESS
      +----------+----------+
  >08 !        MRBSSI       !    SECURITY INFORMATION (QUEUE ADDR)
      +----------+----------+
  >0A !  MRBSOC  !  MRBEC   !    SVC OPERATION CODE
      +----------+----------+       SVC RETURN (ERROR) CODE

                            ABORT I/O VARIANT
      *----------+----------*
  >0C !  MRBABF  !  MRBABL  !    FLAGS
      +----------+----------+       LOGICAL UNIT NUMBER

                            I/O VARIANTS
      *----------+----------*
  >0C !  MRBOC   !  MRBLUN  !    SUB-OPERATION CODE
      +----------+----------+       LOGICAL UNIT NUMBER
  >0E !  MRBSFL  !  MRBUFL  !    SYSTEM FLAGS
      +----------+----------+       REQUESTOR (USER) FLAGS

                            CDE NUMBER WITHIN CDT
      *----------+----------*
  >10 !  FILL02  !  MRBSEG  !    **** RESERVED ****
      +----------+----------+       SEGMENT IDENTIFIER
  >12 !  MRBNAM  !          !    DEVICE NAME
```

```
      +----------+----------+
      /          /          /
      /          /          /
      +----------+----------+
  >1A !        FILL03       !   RESERVED
      +----------+----------+
  >1C !        MRBNUM       !   DEVICE NUMBER
      +----------+----------+
  >1E !        MRBDFL       !   DIOU FLAGS
      +----------+----------+
  >20 !        MRBBUF       !   PARAMETER BUFFER ADDRESS
      +----------+----------+

                            I/O AND IOU VARIANTS
      *----------+----------*
  >10 !        MRBDBA       !   BUFFER ADDR (OFFSET TO BUFFER)
      +----------+----------+
  >12 !        MRBICC       !   INPUT CHAR COUNT / ACTUAL OUTPUT
      +----------+----------+
  >14 !        MRBOCC       !   OUTPUT CHAR COUNT / ACTUAL INPUT
      +----------+----------+

                            DISK I/O VARIANTS
      *----------+----------*
  >16 !        MRBTRK       !   TRACK ADDRESS
      +----------+----------+
  >18 !  MRBSPR  !  MRBSCT  !   SECTORS PER RECORD
      +----------+----------+      SECTOR NUMBER


      *----------+----------*
  >16 !        MRBADU       !   ADU NUMBER
      +----------+----------+
  >18 !        MRBOFF       !   SECTOR OFFSET INTO ADU
      +----------+----------+

                            TERMINAL I/O BLOCK WITH REPLY
      *----------+----------*
  >16 !        MRBRPY       !   REPLY BLOCK ADDRESS (OFFSET)
      +----------+----------+
  >18 !        MRBRES       !   (EXTRA WORD BUFFERED)
      +----------+----------+
  >1A !        MRBRPA       !   REPLY BUFFER ADDR (OFFSET)
      +----------+----------+
  >1C !        MRBRIC       !   REPLY INPUT COUNT FROM REQUESTOR
      +----------+----------+
  >1E !        MRBROC       !   REPLY OUTPUT COUNT FOR REQUESTOR
      +----------+----------+

                            VDT READ BLOCK WITH VDT EXTENSION
      *----------+----------*
  >16 !        MRBROV       !   ZERO, REPLY PTR, OR VALIDATION PTR
      +----------+----------+
   >18 !        MRBXFL         !    EXTENDED REQUEST FLAGS
       +----------+----------+
   >1A ! MRBFCH   !  MRBEVT  !    VDT FILL CHARACTER
       +----------+----------+       VDT EVENT BYTE
   >1C ! MRBCRO   !  MRBCCO  !    VDT CURSOR IN FIELD ROW
       +----------+----------+       VDT CURSOR IN FIELD COLUMN
   >1E ! MRBFRO   !  MRBFCO  !    VDT FIELD BEGINNING ROW
       +----------+----------+       VDT FIELD BEGINNING COLUMN
*

                               WRITE WITH VDT EXTN WITH REPLY
       *----------+----------*
   >20 !        MRBRS3         !    (EXTRA WORD BUFFERED)
       +----------+----------+
   >22 !        MRBRY2         !    REPLY BUFFER POINTER (OFFSET)
       +----------+----------+
   >24 !        MRBRI2         !    REPLY INPUT COUNT FROM REQUESTOR
       +----------+----------+
   >26 !        MRBRO2         !    REPLY OUTPUT COUNT FOR REQUESTOR
       +----------+----------+

                               BASIC FILE I/O BLOCK
       *----------+----------*
   >16 !        MRBRN1         !    RECORD NUMBER FOR REL REC (2 WORDS)
       +----------+----------+
   >18 !                       !
       +----------+----------+

                               KIF I/O BLOCK
       *----------+----------*
   >16 !        MRBCBA         !    CURRENCY BLOCK ADDRESS
       +----------+----------+
   >18 !        MRBRS0         !    RESERVED
       +----------+----------+
   >1A !                       !
       +----------+----------+
   >1C ! MRBCUR   !            !    CURRENCY BLOCK
       +----------+----------+
       /          /          /
       /          /          /
       +----------+----------+

                               I/O UTILITY VARIANT
       *----------+----------*
   >10 ! MRBTYP   !  MRBTFL  !    RESOURCE TYPE
       +----------+----------+       RESOURCE TYPE FLAGS
   >12 !        FILL06         !    RESERVED
       +----------+----------+
   >14 !        FILL07         !    RESERVED
       +----------+----------+
   >16 !        MRBKDB         !    KEY INDEX DEFINITION BLOCK (OFFSET)
       +----------+----------+
   >18 !        MRBRS4         !    RESERVED
```

```
    +----------+----------+
>1A !        MRBFLG       !   UTILITY FLAGS (2 BYTES)
    +----------+----------+
>1C !        MRBDLL       !   DEFINED LOGICAL  RECORD LENGTH
    +----------+----------+
>1E !        MRBDPL       !   DEFINED PHYSICAL RECORD LENGTH
    +----------+----------+
>20 !        MRBPNA       !   PATHNAME ADDR (OFFSET)
    +----------+----------+
>22 !        MRBPRM       !   PARAMETER PTR (OFFSET)
    +----------+----------+
>24 !        MRBRS5       !   RESERVED
    +----------+----------+
>26 !        MRBIFA       !   INITIAL FILE ALLOCATION (2 WORDS)
    +----------+----------+
>28 !                     !
    +----------+----------+
>2A !        MRBSFA       !   SECONDARY FILE ALLOCATION (2 WORDS)
    +----------+----------+
>2C !                     !
    +----------+----------+
>30 SIZE               ** END OF PACKED RECORD


FLAGS FOR FIELD: MRBABF    #0C - FLAGS

  MRFDNC = (X...............) - DO NOT CLOSE
         = (.XXXXXXX........) - RESERVED


FLAGS FOR FIELD: MRBSFL    #0E - SYSTEM FLAGS

  MRFBSY = (X...............) - BUSY
  MRFERR = (.X..............) - ERROR
  MRFEOF = (..X.............) - END OF FILE
  MRFVNT = (...X............) - EVENT CHAR


FLAGS FOR FIELD: MRBUFL    #0F - REQUESTOR (USER) FLAGS

  MRFINT = (X...............) - INITIATE REQUEST
  MRFRPY = (.X..............) - OUTPUT WITH REPLY
  MRFRES = (..X.............) - RESERVED
  MRFACC = (...XX...........) - ACCESS PRIVILEGES
  MRFLOC = (.....X..........) - LOCK/UNLOCK
  MRFOWN = (......XX........) - OWNERSHIP LEVEL


FLAGS FOR FIELD: MRBDFL    #1E - DIOU FLAGS

  MRFLCK = (X...............) - LOCK/UNLOCK
  MRFNAM = (.X..............) - NAME SPECIFIED
```

```
    MRFVRD = (..X.............) - VIRTUAL DEVICE
    MRFWCH = (...X............) - WHICH RELATIVE DEVICE
    MRFREP = (....X...........) - REPLACE


  FLAGS FOR FIELD: MRBXFL    #18 - EXTENDED REQUEST FLAGS

    MRFCSF = (X...............) - CURSOR START OF FIELD DEFN
    MRFNTN = (.X..............) - INTENSITY
    MRFFKR = (..X.............) - BLINKING CURSOR (FLICKER)
    MRFGRA = (...X............) - GRAPHICS DISPLAY (CHAR LT >20)
    MRFEBA = (....X...........) - 8-BIT ASCII
    MRFTER = (.....X..........) - ENABLE TASK EDIT CHAR RETURN
    MRFBP  = (......X.........) - BEEP
    MRFRDB = (.......X........) - RIGHT DISPLAY EDGE BOUNDARY
    MRFCIF = (........X.......) - CURSOR IN-FIELD DEFINED
    MRFFC  = (.........X......) - FILL CHAR DEFINED
    MRFIF  = (..........X.....) - INITIALIZE FIELD
    MRFRFF = (...........X....) - REMAIN IN FULL FIELD
    MRFECO = (............X...) - ECHO
    MRFVRQ = (.............X..) - VALIDATION REQUIRED
    MRFVER = (..............X.) - VERIFICATION ERROR
    MRFWBP = (...............X) - WARNING BEEP


  FLAGS FOR FIELD: MRBTFL    #11 - RESOURCE TYPE FLAGS

           = (XXX.............) - RESERVED
    MRFVD  = (...X............) - VIRTUAL DEVICE
    MRFREM = (....X...........) - REMOTE RESOURCE
    MRFCHN = (.....X..........) - CHANNEL
    MRFDEV = (......X.........) - DEVICE
    MRFFIL = (.......X........) - FILE


  FLAGS FOR FIELD: MRBFLG    #1A - UTILITY FLAGS (2 BYTES)

    MRFFCA = (X...............) - FILE CREATED BY ASSIGN
    MRFFUS = (.XX.............) - FILE USAGE FLAGS
*                                 00=NO SPECIAL USAGE
*                                 01=DIRECTORY FILE
*                                 10=PROGRAM FILE
*                                 11=IMAGE FILE
    MRFSCP = (...XX...........) - LUNO SCOPE
*                                 00=TASK LOCAL
*                                 01=JOB LOCAL
*                                 10=GLOBAL
*                                 11=RESERVED
    MRFGEN = (.....X..........) - AUTOGENERATE LUNO
    MRFACR = (......X.........) - REQUEST AUTOCREATE FILE
    MRFPRM = (.......X........) - 1=MRBPRM VALID (PARMS PRESENT)
    MRFLRL = (........X.......) - USE LOGICAL REC. LENGTH GIVEN
    MRFTMP = (.........X......) - FILE IS TO BE TEMPORARY
    MRFIMW = (..........X.....) - IMMEDIATE WRITE DISK FILES
    MRFDFT = (...........XX...) - DATA FORMAT
*                               00=NORMAL RECORD IMAGE
*                               01=BLANK SUPPRESSED
*                               10,11 RESERVED
    MRFALL = (.............X..) - ALLOCATION MAY GROW
    MRFFTP = (..............XX) - FILE TYPE
*                               00=RESERVED
*                               01=SEQUENTIAL FILE
*                               10=RELATIVE RECORD FILE
*                               11=KEY INDEXED FILE
```

EQUATES:

LABEL	EQUATE TO	VALUE	DESCRIPTION
VARNO	$	>0C	
MRBASZ	$	>0E	ABORT I/O MRB SIZE
MRFVAL	MRFRPY	>01	READ WITH VALIDATION
MRFMDS	MRFLOC	>05	MASTER DO NOT SUSPEND
MRFEXR	MRFOWN	>06	EXTENDED REQUEST
MRFBAD	MRFOWN+1	>07	BLANK ADJ/SET EVENT MODE
MRFWPM	MRFOWN+1	>07	WORD PROCESSING MODE
MRBOSZ	$	>10	
MRBCDE	MRBUFL	>0F	CDE NUMBER WITHIN CDT
MRBSZD	$	>22	SIZE OF DIOU VARIANT
MRBISZ	$	>16	BASIC I/O MRB SIZE
MRBDSZ	$	>1A	DISK I/O MRB SIZE
MRBRSZ	$	>20	BASIC REPLY MRB SIZE
MRBVAL	MRBROV	>16	
MRBXSZ	$	>20	VDT EXTENSION MRB SIZE
MRBRS2	$	>20	NO LONGER USED
MRBVXS	$	>20	READ WITH EXTN/VALIDATION MRB SIZE
MRBXRS	$	>28	WRITE WITH EXTN/REPLY MRB SIZE
MRBFSZ	$	>1A	SIZE OF BASIC FILE I/O MRB
MRBKSZ	$	>30	SIZE OF KIF MRB
MRBUSZ	$	>2E	SIZE OF IOU CALL BLOCK

```
************************************************************
*                                                          *
*  EXTENTION FOR A MAGNETIC   (MTX)             04/13/83   *
*           TAPE DEVICE                                    *
*                                            REV 05/10/83  *
*           LOCATION: SYSTEM AREA                          *
************************************************************
* THE MTX IS AN EXTENSION TO THE PDT USED TO DESCRIBE A
* MAGNETIC TAPE DEVICE.  IT IS USED AS A WORK AREA BY
* THE DSR.


     *----------+----------*
 >00 !  MTXTIL  !          !   TILINE IMAGE
     +----------+----------+
     /          /          /
     /          /          /
     +----------+----------+
 >10 !  MTXSLG  !          !   TILINE IMAGE FOR SYSTEM LOG
     +----------+----------+
     /          /          /
     /          /          /
     +----------+----------+
 >20 !       MTXSVS        !   TILINE UNIT ERROR COUNT
     +----------+----------+
 >22 !       MTXMAJ        !   MAJOR ERROR COUNT
     +----------+----------+
 >24 !       MTXMIN        !   MINOR ERROR COUNT
     +----------+----------+
 >26 !       MTXFLG        !   FLAGS
     +----------+----------+


 FLAGS FOR FIELD: MTXFLG     #26 - FLAGS

   MTFEOT = (X...............) - END-OF-TAPE FLAG
          = (.XXX............) - RESERVED
   MTFODI = (....X...........) - ON-LINE-DIAGNOSTICS
          = (.....XXXXXXXXXXX) - RESERVED
*
```

EQUATES:

LABEL	EQUATE TO	VALUE	DESCRIPTION
MTXSIZ	$	>28	MT PDT + EXTENSION SIZE
MTXCNT	MTXTIL+>	>08	BYTE TRANSFER COUNT
MTXCMD	MTXTIL+>	>0C	TRANSPORT SELECT/COMMAND/ADDR

```
**********************************************************
*      NAME DEFINITION BLOCK  (NDB)             07/16/81  *
*                                                         *
*      LOCATION:  A NAME DEFINITION SEGMENT               *
**********************************************************


      *----------+----------*
  >00 !       NDBNDB        !   FIXED LINK - SEQ PROCESSING
      +----------+----------+
  >02 !       NDBPAR        !   POINTER TO PARENT NDB
      +----------+----------+
  >04 !       NDBLLL        !   POINTER TO LEFT SON
      +----------+----------+
  >06 !       NDBRRR        !   POINTER TO RIGHT SON
      +----------+----------+
  >08 !       NDBNAM        !   PTR TO THE NAME
      +----------+----------+
  >0A !       NDBSVB        !   ANCHOR OF STAGE VALUES
      +----------+----------+
  >0C !  NDBBAL  !  NDBWAT  !   BALANCE FACTOR
      +----------+----------+      SON INDICATOR

  EQUATES:

     LABEL   EQUATE TO   VALUE   DESCRIPTION
     -----   ---------   -----   ------------------------------
     NDBSIZ   $              >0E
```

```
************************************************************
*                                                          *
*    NAME DEFINITION SEGMENT OVERHEAD  (NDS)     12/16/81  *
*                                                          *
*    LOCATION:  A NAME DEFINITION SEGMENT                  *
************************************************************
```

```
      *----------+----------*
  >00 !       NDSHED        !   1ST ENTRY ON FREE MEMORY LIST
      +----------+----------+
  >02 !       NDSLNK        !   PTR TO FREE MEMORY CHAIN
      +----------+----------+
  >04 !       NDSRES        !   RESERVED TABLE AREA BOUNDRY
      +----------+----------+
  >06 !       NDSEND        !   ACTUAL ADDRESS OF END OF SEG
      +----------+----------+
  >08 !       NDSUSE        !   CURRENT MEMORY USAGE
      +----------+----------+
  >0A !       NDSHI         !   HIGHEST MEMORY ALLOCATION
      +----------+----------+
  >0C !       NDSJSB        !   PTR TO JSB OR SSB OF OWNER
      +----------+----------+
  >0E !  NDSNUL  !  NDSOWN  !   HANDY NULL STRING
      +----------+----------+      SEGMENT IN USE IF NON-ZERO
  >10 !       NDSSTR        !   PTR TO ROOT OF SYN TREE
      +----------+----------+
  >12 !       NDSLTR        !   PTR TO ROOT OF LGN TREE
      +----------+----------+
  >14 !       NDSSDB        !   PTR TO 1ST SDB FOR THE JOB
      +----------+----------+
  >16 !       NDSSYN        !   FIXED LINK OF SYNONYM NDBS
      +----------+----------+
  >18 !       NDSLGN        !   FIXED LINK OF NAME NDBS
      +----------+----------+
  >1A !       NDSTMP        !   TEMPORARY PACKET ADDRESS
      +----------+----------+
```

EQUATES:

LABEL	EQUATE TO	VALUE	DESCRIPTION
NDSSIZ	$	>1C	LENGTH OF NDS OVERHEAD

```
*************************************************************
*                                                           *
* SYSTEM CRASH CODE EQUATES   (NFCRSH)           06/08/83   *
*                                                           *
*************************************************************
* NFCRSH LISTS ALL POSSIBLE SYSTEM CRASH CODES GENERATED BY
* DNOS.  SOME CODES ARE ALSO RESERVED AS THEY ARE USED BY
* DX10 AND USE OF THOSE BY DNOS WOULD NOT BE DESIRABLE.
*
CSH00E EQU  >000E             PMTLDR - CANNOT ASSIGN TO ROLL
*                                      FILE
*CSH010 THRU CSH012           RESERVED-USED BY DX10
* CSH013 THRU CSH01F  ILLEGAL INTERRUPT AT LEVEL 3 THRU F
*CSH020                       RESERVED-DX10-ILLEGAL INTERNAL
*                                      INTERRUPT
CSH021 EQU  >0021             PMUMGR - INCONSISTENT STRUCTURE
CSH022 EQU  >0022             NFTMGR - INCONSISTENT STRUCTURE
CSH023 EQU  >0023             NFSCHD - QUEUING ERROR
CSH024 EQU  >0024             IOBM   - INCONSISTENT STRUCTURE
CSH025 EQU  >0025             ILLEGAL SYSTEM XOP
CSH026 EQU  >0026             PMROLL - CANNOT EXTEND SWAP FILE
CSH027 EQU  >0027             PMROLL - SWAP FILE WRITE ERROR
CSH028 EQU  >0028             PMLDSG - SWAP FILE READ ERROR
CSH029 EQU  >0029             NFPOP - UNEXPECTED ERROR RETURNED
CSH02A EQU  >002A             NUCLEUS - INCONSISTENT STRUCTURE
*CSH02B                       RESERVED-DX10-ERR IN LDT BUILT
*                                      FOR PROG. FILE
CSH02C EQU  >002C             NFENAB - SCHEDULER INHIBIT NEG.
*CSH02D                       RESERVED-DX10-TM$LDR TOOK END
*                                      ACTION
*CSH02E                       RESERVED-DX10-SO$CPR ERROR
CSH02F EQU  >002F             SYSTEM OVERLAY LOAD ERROR
CSH030 EQU  >0030             NFTMGR - NO SYSTEM TABLE AREA
*CSH031                       RESERVED-DX10-UNEXP ERR RETURN
*                                      IN RM$REL
*CSH032                       RESERVED-DX10-UNEXP ERR RETURN
*                                      IN BM$TRW
*CSH033                       RESERVED-DX10-UNEXP ERR RETURN
*                                      IN BM$W
*CSH034                       RESERVED-DX10-UNEXP ERR RETURN
*                                      IN BM$CLO
*CSH035                       RESERVED-DX10-UNEXP ERR RETURN
*                                      IN BM$FLS
*CSH036                       RESERVED-DX10-UNEXP ERR RETURN
*                                      IN BM$SCH
*CSH040 THRU CSH045           RESERVED-DX10-
CSH046 EQU  >0046             SEGMGR - INCONSISTENT STRUCTURE
CSH048 EQU  >0048             JOBMGR - END ACTION TAKEN
CSH04A EQU  >004A             JOBMGR - TASK QUEUING ERROR
CSH04B EQU  >004B             JOBMGR - ERROR FROM SEG MGR
CSH04C EQU  >004C             JOBMGR - ERROR FROM IOU
CSH04D EQU  >004D             JOBMGR - CANNOT GET TABLE AREA
```

```
*CSH050                       RESERVED-DX10-
CSH051 EQU  >0051             PROGRAM FILE SVC'S -
*                                INCONSISTENT LDT LIST
CSH060 EQU  >0060             NFINT2 - INTERNAL INTERRUPT
*                             60 THRU 6F RESERVED FOR
*                             INTERNAL INTERRUPTS 0 - F
*                             60 - INVALID INTERNAL INTERRUPT
*                             61 - MEMORY PARITY
*                             62 - ILLEGAL INSTRUCTION
*                             63 - TILINE TIMEOUT
*                             64 - ILLEGAL SUPERVISOR CALL
*                                (RESERVED,SOFTWARE DETECTED)
*                             65 - MAPPING ERROR
*                             66 - PRIVILEGED OPCODE
*                             67 - TASK IS BEING KILLED
*                                (RESERVED,SOFTWARE DETECTED)
*                             68 - NOT ENOUGH USER TASK AREA
*                                   SOFTWARE DETECTED
*                             69 - SEGMENT NOT PRESENT
*                             6A - EXECUTE PROTECT VIOLATION
*                             6B - WRITE PROTECT VIOLATION
*                             6C - STACK OVERFLOW
*                             6D - HARDWARE BREAKPOINT
*                             6E - 12 MS CLOCK EXPIRED
*                             6F - ARITHMETIC OVERFLOW
*                                       DISK SPACE
*CSH070 THRU CSH076           RESERVED-DX10-
CSH077 EQU  >0077             MEDIA CHANGE OCCURRED ON SYS DISK
CSH080 EQU  >0080             DSKMGR - END ACTION TAKEN
*CSH081                       RESERVED-DX10-
CSH082 EQU  >0082             DSKMGR - UNDEFINED OP CODE
CSH083 EQU  >0083             DSKMGR - ADU ALLOCATED ALREADY USED
CSH084 EQU  >0084             DSKMGR - FIRST AVAILABLE ADU
*                                      OUT OF RANGE
CSH085 EQU  >0085             DSKMGR - ILLEGAL PARTIAL BIT MAP
*                                      NUMBER REQUESTED
CSH086 EQU  >0086             DSKMGR - CACHED BIT MAP HAS
*                                      BEEN MODIFIED
CSH087 EQU  >0087             DSKMGR - READ AFTER WRITE OF
*                             PARTIAL BIT MAP DOES NOT VERIFY
*CSH088 THRU CSH089           RESERVED-DX10-
CSH090 EQU  >0090             INVALID USE OF VT01
CSH094 EQU  >0094             DSR940 - CAN'T GET BUFFER TABLE AREA
CSH0A0 EQU  >00A0             FILMGR - END ACTION TAKEN
CSH0A1 EQU  >00A1             FILMGR - ERROR LOADING FM OVERLAY
CSH0A2 EQU  >00A2             FILMGR-INCONSISTENT STRUCTURE
*CSH0A3 THRU CSH0A4           RESERVED-DX10-
*CSH0AF                       RESERVED-DX10-
CSH0B0 EQU  >00B0             NAMMGR - END ACTION TAKEN
CSH0B1 EQU  >00B1             NAMMGR - PASCAL RUN-TIME ABORT
CSH0B2 EQU  >00B2             IOTBID - END ACTION TAKEN
CSH0B3 EQU  >00B3             IPCTSK - END ACTION TAKEN
CSH0B5 EQU  >00B5             PMOVYL - END ACTION TAKEN
```

```
CSH0B6 EQU  >00B6             PMTBID - END ACTION TAKEN
CSH0B7 EQU  >00B7             PMWRIT - END ACTION TAKEN
CSH0B8 EQU  >00B8             PMTLDR - END ACTION TAKEN
CSH0B9 EQU  >00B9             PMTERM - END ACTION TAKEN
CSH0BA EQU  >00BA             PMSBUF - END ACTION TAKEN
CSH0BB EQU  >00BB             PMRWTK - END ACTION TAKEN
CSH0BD EQU  >00BD             PMSBID - END ACTION TAKEN
CSH0BE EQU  >00BE             RCP - END ACTION TAKEN
CSH0C0 EQU  >00C0             NFEOBR - TSBIO HAS BECOME NEGATIVE
*CSH0E0 THRU CSH0E5           RESERVED-DX10-
CSH100 EQU  >0100             IOU - END ACTION TAKEN
CSH101 EQU  >0101             IOU - WRONG SEGMENT MAPPED
CSH102 EQU  >0102             IOU - LOOKUP, DE-LINK FAILURE
*CSH103 THRU CSH106           RESERVED-DX10-
CSH107 EQU  >0107             IOU - BAD FILE LDT LIST
*CSH108 THRU CSH109           RESERVED-DX10-
CSH10A EQU  >010A             IOU - ERROR RETURNING ADU
*                                   JUST OBTAINED
*CSH10B THRU CSH10D           RESERVED-DX10-
CSH10E EQU  >010E             IOU - FCB BLOCK COUNT OVERFLOW
*CSH120 THRU CSH123           RESERVED-DX10-
*CSH130 THRU CSH131           RESERVED-DX10-
CSH132 EQU  >0132             RPUTIL - END ACTION TAKEN
*CSH133 THRU CSH137           RESERVED-DX10-
CSH138 EQU  >0138             RPIV -BIT MAP TABLE ERROR
CSH139 EQU  >0139             RPINV2 - DISK ALLOCATION FAILURE
CSH13A EQU  >013A             RPINV2 - BAD BIT MAP NUMBER
CSH13B EQU  >013B             RPINV2 - BAD ADU LIST RANGE OVERLAP
CSH13C EQU  >013C             NFPWUP - NO POWER DOWN INTERRUPT
CSH13D EQU  >013D             NFPWUP - CANNOT FIND RTWP CONTEXT
CSH13E EQU  >013E             NFPWUP - INVALID RTWP CONTEXT
*CSH13F                       RESERVED-DX10-
*CSH140                       RESERVED-DX10-
CSH141 EQU  >0141             RESERVED-DX7 -NO POWER FAIL
*                                      RECOVERY SUPPORT
CSH142 EQU  >0142             UNIV BLD -NO TERMINAL AVAILABLE
CSH143 EQU  >0143             UNIV BLD -I/O ERROR TO TERMINAL
*                                       WHILE BUILDING DISK
CSH144 EQU  >0144             UNIV BLD -NO RESPONSE TO INITIAL
*                                       MESSAGE
CSH145 EQU  >0145             IPC - INCONSISTENT DATA STRUCTURES
CSH146 EQU  >0146             DIOU TOOK END ACTION
*CSH147                       RESERVED-DX10-PDT'S POINTER TO
*                                      PRB IS INVALID
*CSH150                       RESERVED-DX7 -DISK CHANGED WITH
*                                      NO UNLOAD (UV) COMMAND
CSH160 EQU  >0160             (RESERVED DX10-TM$BID END ACTION)
CSH161 EQU  >0161             IOU (SECMGR) - UNABLE TO OPEN
*                                  LUNO TO .S$CLF
CSH162 EQU  >0162             IOU (SECMGR) - UNABLE TO CREATE
*                                  OR MAP SPECIAL SEGMENT FOR
*                                  BUILDING CAPABILITY LIST
CSH163 EQU  >0163             RESTART - JUST MADE CRASH FILE
```

```
*                                 BIGGER, SO WE FORCED CRASH
*CSH177                       RESERVED-DX10-
*************************************************************
*                                                           *
*      SYSTEM LOADER FLASH CRASH CODES                      *
*                                                           *
*************************************************************
FLSH01 EQU  >0001             LOAD DEVICE I/O ERROR
FLSH02 EQU  >0002             NOT ENOUGH PHYSICAL MEMORY
FLSH03 EQU  >0003             CAN'T FIND SYSTEM DISK PDT
FLSH04 EQU  >0004             ERROR IN PROG FILE DIRECTORY
FLSH05 EQU  >0005             S$IPL INCONSISTENT WITH REV.
*                               LEVEL OF CURRENT SYSTEM
FLSH06 EQU  >0006             ERROR IN DM BIT MAP ROUTINE
FLSH08 EQU  >0008             CAN'T FIND SYSTEM LOADER FILE
FLSH09 EQU  >0009             CAN'T FIND KERNEL PROGRAM FILE
FLSH0A EQU  >000A             CAN'T FIND A SYSTEM SEGMENT
FLSH0B EQU  >000B             NO PATCHES APPLIED TO SYSTEM
FLSH0C EQU  >000C             SOFTWARE VERSION TOO OLD
FLSH0D EQU  >000D             CAN'T FIND UTILITIES PROG FILE
FLSH0E EQU  CSH00E            CAN'T FIND SYSTEM ROLL FILE
FLSH0F EQU  >000F             KERNEL FILE LEVEL INCONSISTENT
*                                WITH UTILITY FILE
FLSH11 EQU  >0011             CAN'T GET SYSTEM TABLE AREA
FLSH13 EQU  >0013             LOGICAL ADDRESS OVERFLOW
FLSH14 EQU  >0014             CAN'T LOAD WCS FILE
*FLSH60-6F  EQU  >60->6F      INTERNAL INTERRUPT (LEVEL 2)
FLSH68 EQU  >0068             NOT ENOUGH USER TASK AREA
```

```
************************************************************
*                                                          *
*  TASK STATE CODES           (NFSTAT)          09/20/83   *
*                                                          *
*  NOTES:                                                  *
*      1) THIS MODULE REQUIRES NFER00 THRU NFER40 BE       *
*         COPIED ALSO.                                     *
*                                                          *
*      2) CHANGES TO THIS MODULE REQUIRE CORRESPONDING     *
*         CHANGES IN 3 OF THE MESSAGES IN THE SVC MESSAGES *
*         FILE.   (SVC >35, SVC >07, AND SVC 0E)           *
************************************************************
* THESE EQUATES DESCRIBE ALL THE LEGAL TASK STATE CODES AND
* JOB STATE CODES USED BY THE OS.
TSTACT EQU  BYTE00           TASK IS ON ACTIVE LIST
TSTWOM EQU  BYTE01           TASK IS WAITING ON MEMORY
TSTJWT EQU  BYTE02           JOB IN A NONEXECUTABLE STATE
TSTLIP EQU  BYTE03           TASK LOAD IN PROGRESS
TSTTRM EQU  BYTE04           TASK HAS TERMINATED
TSTDLY EQU  BYTE05           TASK IS IN A TIME DELAY
TSTSUS EQU  BYTE06           TASK UNCONDITIONALLY SUSPENDED
TSTENX EQU  BYTE07           WAITING FOR TEN X PROCESSOR
*                            COMPLETION
TSTSIO EQU  BYTE09           TASK SUSPENDED FOR I/O
TSTSAI EQU  BYTE0F           SUSPENDING FOR ABORTING I/O
TSTOVL EQU  BYTE14           WAITING FOR OVERLAY LOAD SVC
TSTCOA EQU  BYTE17           TASK AWAITING COROUTINE ACT
TSTWIO EQU  BYTE19           WAITING FOR INITIATED I/O
TSTDOR EQU  BYTE1E           WAITING FOR DOOR TO OPEN
TSTSBT EQU  BYTE1F           WAITING FOR SCHD TASK BID SVC
TSTIV  EQU  BYTE20           WAITING FOR INSTALL VOLUME SVC
TSTDMG EQU  BYTE22           WAITING FOR DISK MGR SVC
TSTQIN EQU  BYTE24           AWAITING QUEUE INPUT
TSTIT  EQU  BYTE25           WAITING FOR INSTALL TASK SVC
TSTIP  EQU  BYTE26           WAITING FOR INSTALL PROC SVC
TSTIO  EQU  BYTE27           WAITING FOR INSTALL OVLY SVC
TSTDT  EQU  BYTE28           WAITING FOR DELETE TASK SVC
TSTDP  EQU  BYTE29           WAITING FOR DELETE PROC SVC
TSTDO  EQU  BYTE2A           WAITING FOR DELETE OVLY SVC
TSTBID EQU  BYTE2B           TASK SUSPENDED FOR BID SVC
TSTRWT EQU  BYTE2D           WAITING FOR READ/WRITE TSK SVC
TSTWOT EQU  BYTE30           WAITING FOR SYSTEM TABLE AREA
TSTMNI EQU  BYTE31           WAITING FOR MAP PROG NAME TO ID
TSTUV  EQU  BYTE34           WAITING FOR UNLOAD VOLUME SVC
TSTAIO EQU  BYTE36           WAITING FOR ANY I/O
TSTAPS EQU  BYTE37           WAITING FOR ASG PROG FILE SPACE
TSTINV EQU  BYTE38           WAITING FOR INIT NEW VOL SVC
TSTSEM EQU  BYTE3D           TASK SUSPENDED FOR SEMAPHORE
TSTSEG EQU  BYTE40           TASK AWAITING SEG MGR SERVICES
TSTEWT EQU  BYTE42           WAITING FOR EVENT COMPLETION
TSTNMG EQU  BYTE43           WAITING FOR NAME MGR SVC
TSTJMR EQU  BYTE48           TASK WAITING ON JOB MGR SVC
```

```
TSTFRL EQU  BYTE4A           WAITING FOR FORCED ROLL SVC
TSTRCP EQU  BYTE4C           WAITING FOR RETURN CODE PROC
*************************************************************
*                                                         *
*      JOB STATE CODES                                    *
*                                                         *
*************************************************************
JSTCRE EQU  BYTE01           JOB IS BEING CREATED
JSTEXC EQU  BYTE02           JOB IS IN A EXECUTABLE STATE
JSTHLT EQU  BYTE03           JOB IS HALTED
JSTTRM EQU  BYTE04           JOB IS TERMINATING
JSTEXP EQU  BYTE05           JCA IS BEING EXPANDED
```

```
***********************************************************
*                                                         *
*    NAME REQUEST BLOCK  (NRB)                   09/09/83 *
*                                                         *
*    LOCATION:  SYSTEM TABLE AREA                         *
***********************************************************

                                NAME MGR REQUEST BLOCK
      *----------+----------*
  >00 !  NRBSOC  !  NRBEC   !   SVC CODE
      +----------+----------+      ERROR CODE
  >02 !  NRBOC   !  NRBFLG  !   SUBOPCODE
      +----------+----------+      USER SET FLAGS
  >04 !       NRBNAM        !   PTR TO "NAME" (OR NAME LIST)
      +----------+----------+
  >06 !       NRBVAL        !   PTR TO "VALUE" OR PATHNAME
      +----------+----------+
  >08 !       NRBPRM        !   PTR TO "PARMS" LIST
      +----------+----------+


      *----------+----------*
  >0A !       NRBPNO        !   PATHNAME NUMBER
      +----------+----------+
  >0C !       NRBRSV        !   RESERVED
      +----------+----------+
  >0E !  NRBTSK  !  NRBSTG  !   TASK ID
      +----------+----------+      STAGE NUMBER
  >10 !       NRBSMT        !   SMT FOR NAME SEGMENT
      +----------+----------+
  >12 !       NRBSSB        !   SSB FOR NAME SEGMENT
      +----------+----------+
  >14 !  NRBVBL  !  NRBPBL  !   VALUE BUFFER LENGTH
      +----------+----------+      PARMS BUFFER LENGTH
  >16 !       NRBLNA        !   NAME POINTER -LOGICAL ADDRESS
      +----------+----------+
  >18 !       NRBLVA        !   VALUE POINTER-LOGICAL ADDRESS
      +----------+----------+
  >1A !       NRBLPA        !   PARMS POINTER-LOGICAL ADDRESS
      +----------+----------+


      *----------+----------*
  >0A !       NRBSSZ        !   SEGMENT SIZE
      +----------+----------+


      *----------+----------*
  >0A !       NRBSPF        !   SUCCESSOR/PREDECESSOR FLAG
      +----------+----------+
```

FLAGS FOR FIELD: NRBFLG #03 - USER SET FLAGS

```
NRFLOG = (X...............) - LOG = 1, ELSE SYN OPERATION
NRFINT = (.X..............) - INITIAL TASK IN JOB IF TRUE
NRFBID = (..X.............) - BID = 1, TERMINATION = 0
NRFGLO = (...X............) - GLOBAL REQUEST = 1
NRFRID = (....X...........) - RUN ID SPECIFIED = 1
NRFNMX = (.....X..........) - NAME SEGMENT CANNOT EXPAND
NRFPRO = (......X.........) - PROTECT NAME
NRF007 = (.......X........) - FLAG BIT 7 UNUSED
```

EQUATES:

LABEL	EQUATE TO	VALUE	DESCRIPTION
NRBVAR	$	>0A	
NRBSID	$	>0C	SEGMENT ID
NRBSIZ	$	>1C	SIZE OF BASIC BLOCK TO BUFFER

```
************************************************************
*                                                          *
*  OVERLAY AREA DESCRIPTOR    (OAD)              12/07/79  *
*                                                          *
*            LOCATION: WITH SYSTEM OVERLAY AREAS           *
************************************************************
* THE OAD PRECEDES A SYSTEM OVERLAY AND DESCRIBES ITS SIZE
* AND LOCATION.

                                   SIZE OF OVERLAY DESCRIPTOR
      *----------+----------*
>FFF0 !        OADSMT       !   SMT ADDRESS OF OVERLAY AREA SEGMENT
      +----------+----------+
>FFF2 !        OADSSB       !   SSB ADDRESS OF OVERLAY AREA SEGMENT
      +----------+----------+
>FFF4 !        OADOFF       !   OFFSET INTO SEGMENT OF OVERLAY CODE
      +----------+----------+
>FFF6 !        OADBSZ       !   NUMBER OF BYTES TO READ
      +----------+----------+
>FFF8 !        OADSIZ       !   SIZE OF OVERLAY AREA
      +----------+----------+
>FFFA !        OADOVN       !   CURRENT OVERLAY IN AREA -1=NONE
      +----------+----------+
>FFFC !        OADUSE       !   NUMBER OF TASKS USING THE OVERLAY
      +----------+----------+
>FFFE !        OADOAD       !   POINTER TO NEXT OVERLAY AREA
      +----------+----------+


  EQUATES:

     LABEL   EQUATE TO   VALUE   DESCRIPTION
     -----   ---------   -----   ------------------------------
     OADSZ   $              >FFF0    SIZE OF OVERLAY DESCRIPTOR
```

```
*************************************************************
*                                                           *
*  OVERLAY AREA WAIT BLOCK    (OAW)              11/08/79  *
*                                                           *
*            LOCATION: SYSTEM TABLE AREA                    *
*************************************************************
* THE OAW IS USED TO REPRESENT THE TASK WAITING FOR ONE
* SYSTEM OVERLAY OF A POOL OF OVERLAYS.  THE POOL MANAGER
* MAINTAINS A LIST OF OAW ENTRIES AND WHEN AN OVERLAY IS
* FREE, CHECKS TO SEE IF ANY TASK IS WAITING FOR IT.  IF SO,
* THE OVERLAY IS LOADED AND TH TASK IS ACTIVATED.


      *----------+----------*
  >00 !        OAWOAW       !   NEXT WAIT BLOCK
      +----------+----------+
  >02 !        OAWJSB       !   JSB OF WAITING TASK
      +----------+----------+
  >04 !        OAWTSB       !   TSB OF WAITING TASK
      +----------+----------+
  >06 !        OAWOVN       !   NUMBER OF OV AREA BEING WAITED FOR
      +----------+----------+
```

EQUATES:

LABEL	EQUATE TO	VALUE	DESCRIPTION
OAWSIZ	$	>08	

```
************************************************************
*                                                          *
*  OWNED SEGMENT ENTRY        (OSE)             09/23/81   *
*                                                          *
*             LOCATION: JCA                                *
************************************************************
* THE OSE DESCRIBES A SEGMENT WHICH IS EXCLUSIVELY USED BY A
* TASK.  IT IS LINKED TO THE TSB.


      *----------+----------*
  >00 !        OSEOSE       !   LINK TO NEXT OWNED SEGMENT ENTRY
      +----------+----------+
  >02 !        OSESSB       !   SSB ADDRESS OF OWNED SEGMENT
      +----------+----------+
  >04 !        OSESMT       !   SSB ADDRESS OF SEGMGR TABLE AREA
      +----------+----------+

  EQUATES:

     LABEL   EQUATE TO   VALUE   DESCRIPTION
     -----   ---------   -----   ------------------------------
     OSESIZ  $              >06
```

```
***********************************************************
*                                                         *
*  OPENING TASK IDENTIFIER     (OTI)            08/25/81  *
*                                                         *
*          LOCATION:  JCA OR STA                          *
***********************************************************
*  THE OTI IS AN ELEMENT OF A SINGLY LINKED LIST CONTAINING
*  TSB  ADDRESSES OF TASKS WHICH HAVE OPENED THE LUNO
*  ASSOCIATED WITH THE PARENT LDT.


      *----------+----------*
  >00 !       OTIOTI        !   LINK TO NEXT OTI
      +----------+----------+
  >02 !       OTITSB        !   TSB ADDRESS OF OPENING TASK
      +----------+----------+


  EQUATES:

     LABEL   EQUATE TO   VALUE   DESCRIPTION
     -----   ----------  -----   ------------------------------
     OTISIZ   $-OTIOTI      >04    SIZE OF OTI
```

```
************************************************************
*                                                          *
*  OVERHEAD BEET        (OVB)                   10/04/83   *
*                                                          *
*           LOCATION: USER MEMORY                          *
************************************************************
* THE OVB IS THE 32 BYTES PRECEDING A SEGMENT WHEN IT IS IN
* MEMORY.  THE OVB INCLUDES LINKAGE, TYPE, AND STATUS
* INFORMATION ABOUT THE SEGMENT.
*
* THE OVFROL FLAG ALSO HAS THE MEANING "SEGMENT LOGICALLY
* NOT IN MEMORY".
* THE OVBTIM IS THE COUNT OF TASKS WHICH BOTH:
*  A) HAVE THE SEGMENT MAPPED IN OR LOADED, AND
*  B) HAVE ALL THEIR MAPPED OR LOADED SEGMENTS PHYSICALLY
*     IN MEMORY.
* AN OVERRUN BEET IS ALLOCATED AT THE END OF THE SEGMENT
* TO PREVENT PROBLEMS WHICH COULD OCCUR BECAUSE OF /12 CPU
* PRE-FETCH OR CACHE FLUSH.
*


      *----------+----------*
  >00 !        OVBLEN       !   LENGTH OF SEGMENT + OVERHEAD BEET
      +----------+----------+      + OVERRUN BEET (BEETS)
  >02 !        OVBPTR       !   TSB ADDRESS WHEN BLOCK IS ON TOL
      +----------+----------+      LDT ADDRESS IF BUFFER SEGMENT
  >04 !        OVBFLK       !   FORWARD LINK TO NEXT BLOCK
      +----------+----------+
  >06 !        OVBBLK       !   BACKWARD LINK TO NEXT BLOCK
      +----------+----------+
  >08 !  OVBTYP  !  OVBIOC  !   SEGMENT TYPE
      +----------+----------+      TILINE AND 911 I/O OUTSTANDING
  >0A !        OVBJSB       !   JSB POINTER CORRESPONDING TO OVBPTR
      +----------+----------+
  >0C !        OVBSSB       !   SEGMENT STATUS BLOCK POINTER
      +----------+----------+
  >0E !        OVBSMT       !   TABLE AREA SSB ADDRESS
      +----------+----------+
  >10 !        OVBQLK       !   QUEUE LINK (DEALLOCATE/WRITE Q)
      +----------+----------+
  >12 !        OVBBRB       !   POINTER TO FORCE WRITE BRB
      +----------+----------+
  >14 !  FILL00  !          !   RESERVED
      +----------+----------+
  >16 !  OVBSTS  !  FILL01  !   SEGMENT STATUS (IN MEMORY STATUS)
      +----------+----------+      RESERVED
  >18 !        OVBEXC       !   EXECUTION TIME SINCE LOAD
      +----------+----------+
  >1A !  FILL03  !          !   RESERVED FOR FUTURE USE
      +----------+----------+
  >1C !          !          !
```

```
    +----------+----------+
>1E !        OVBTIM       !   TASK IN MEMORY COUNT
    +----------+----------+
```

```
FLAGS FOR FIELD: OVBSTS    #16 - SEGMENT STATUS (IN MEMORY STATUS)

  OVFWRT = (X...............) - SEGMENT ON WRITE QUEUE
  OVFROL = (.X..............) - FORCE ROLL THIS SEGMENT
  OVFUBS = (..X.............) - USER SEG USED AS FILE BUFFER
```

EQUATES:

LABEL	EQUATE TO	VALUE	DESCRIPTION
OVSHDR	>FF	>FF	LIST HEADER = -1
OVSDAT	>00	>00	DATA FILE SEGMENT = 0
OVSPRO	>01	>01	PROGRAM FILE SEGMENT = 1
OVSMEM	>02	>02	MEMORY BASED SEGMENT = 2
OVSFRE	>03	>03	FREE BLOCK = 3
OVSDEL	>04	>04	DEALLOCATE QUEUE SEGMENT = 4
OVBSIZ	$	>20	

```
*************************************************************
*                                                           *
*  OVERLAY TABLE ENTRY      (OVT)                05/10/79   *
*                                                           *
*           LOCATION: IN RPSDAT                             *
*************************************************************
*  THE SYSTEM OVERLAY TABLE (SOV) CONSISTS OF A NUMBER OF
*  OVERLAY TABLE ENTRIES (OVT).  IT IS BUILT DURING SYSTEM
*  GENERATION, BASED ON THE CHOICES REQUESTED THEN.

      *----------+----------*
  >00 !       OVTREC        !   RECORD NUMBER OF OVERLAY IMAGE
      +----------+----------+
  >02 !       OVTSIZ        !   SIZE OF OVERLAY CODE
      +----------+----------+
  >04 !       OVTLOD        !   NATURAL LOAD ADDRESS OF OVERLAY
      +----------+----------+
************************************************************
*                                                          *
*  PARTIAL BIT MAP              (PBM)          03/06/80    *
*                                                          *
*           LOCATION: DISK, PARTIAL BIT MAP TABLE          *
************************************************************
* THE PBM DESCRIBES THE CURRENT ALLOCATION OF A DISK.  THE
* IN-MEMORY PBM IS IN THE PARTIAL BIT MAP TABLE SET ASIDE
* FOR ONLY PBM TABLES.  EACH TIME A FILE IS CREATED OR
* EXTENDED, THE PBM ISUPDATED IN BOTH MEMORY AND DISK
* REPRESENTATIONS.


      *----------+----------*
  >00 !  PBMMAX  !  PBMNUM  !    NUMBER OF PARTIAL MAPS
      +----------+----------+       PARTIAL MAP NUMBER IN BUFFER
  >02 !       PBMFAU        !    FIRST AVAILABLE ADU ON DISK
      +----------+----------+
  >04 !       PBMLRC        !    LRC CHECKSUM OF MEMORY PBM
      +----------+----------+
  >06 !       PBMLCB        !    ADU OF LARGEST CONTIGUOUS BLK
      +----------+----------+
  >08 !  PBMMAP  !          !    PARTIAL BIT MAP
      +----------+----------+
      /          /          /
      /          /          /
      +----------+----------+
*
*  THIS SECTION MAPS EACH PBM; THE DISK MANAGER CAN EXAMINE
*  THIS "MAP" TO DETERMINE ON A FIRST-FIT BASIS THE PARTIAL
*  BIT MAP FROM WHICH TO ALLOCATE WITHOUT SEQUENTIALLY
*  SEARCHING THE DISK-RESIDENT BIT MAPS.  THE FOLLOWING
*  STRUCTURE IS REPEATED AS REQUIRED FOR EACH PARTIAL BIT MAP:
*


      *----------+----------*
  >00 !       PBMBIG        !    LARGEST CONTIGUOUS BLK IN PBM
      +----------+----------+
  >02 !       PBMBGN        !    CONTIGUOUS BLK AT START
      +----------+----------+
  >04 !       PBMEND        !    CONTIGUOUS BLK AT END
      +----------+----------+


  EQUATES:

     LABEL   EQUATE TO   VALUE   DESCRIPTION
     -----   ---------   -----   ------------------------------
     PBMBLK   256          >100    PARTIAL BIT MAP BLOCK SIZE
     PBMLEN   PBMBLK/2     >7F0    ADU'S IN PARTIAL BIT MAP
     PBMTAB   $            >106    START OF BIT MAP TABLE
```

```
PBMSIZ   $             >106   PBM BUFFER/TABLE SIZE
PBMTES   $             >06   SIZE OF EACH TABLE ENTRY
```

```
***********************************************************
*                                                         *
*  PHYSICAL DEVICE TABLE     (PDT)             06/22/83 *
*                                                         *
*          LOCATION: SYSTEM AREA                          *
***********************************************************
* EACH DEVICE GENERATED INTO A SYSTEM IS REPRESENTED BY A
* PDT.  THE PDT IS USED AS A WORK AREA FOR THE DEVICE SERVICE
* ROUTINE WHILE PROCESSING REQUESTS FOR THE PARTICULAR DEVICE.
************************************************************


      *----------+----------*
  >00 !        PDTPDT       !   FORWARD LINKAGE TO NEXT PDT
      +----------+----------+
  >02 !  PDTNAM  !          !   DEVICE NAME
      +----------+----------+
  >04 !          !          !
      +----------+----------+
  >06 !        PDTNUM       !   DEVICE NUMBER
      +----------+----------+
  >08 !  PDTLC   !  PDTIL   !   LUNOS ASSIGNED COUNT
      +----------+----------+      INITIATE REQUEST LIMIT
  >0A !  PDTCHR  !  PDTCDT  !   BID CHARACTER
      +----------+----------+      CDT NUMBER
  >0C !        PDTCDE       !   DEVICE CDE MASK
      +----------+----------+
  >0E !        PDTFLG       !   DEVICE STATUS FLAGS EXTENSION
      +----------+----------+
  >10 !        PDTMAP       !   DSR MAP FILE
      +----------+----------+
      /          /          /
      /          /          /
      +----------+----------+
  >1C !        PDTJOB       !   JSB ADDRESS OWNER JOB
      +----------+----------+
  >1E !        PDTRPB       !   RESOURCE PRIVILEGE BLOCK POINTER
      +----------+----------+
  >20 !        PDTBQ        !   BID REQUEST QUEUE
      +----------+----------+
  >22 !        PDTR0        !   R0  - DSR SCRATCH
      +----------+----------+
  >24 !        PDTPRB       !   R1  - QUEUED PRB ADDRESS
      +----------+----------+
  >26 !        PDTDSF       !   R2  - DEVICE STATUS FLAGS
      +----------+----------+
  >28 !  PDTDTF  !  PDTTYP  !   R3  - DEVICE TYPE FLAGS
      +----------+----------+      - DEVICE TYPE
  >2A !        PDTDIB       !   R4  - DEVICE INFO BLOCK ADDRESS
      +----------+----------+
  >2C !        PDTR5        !   R5  - DSR SCRATCH
      +----------+----------+
>2E !         PDTR6         !   R6  - DSR SCRATCH
    +----------+----------+
>30 !         PDTR7         !   R7  - DSR SCRATCH
    +----------+----------+
>32 !         PDTR8         !   R8  - DSR SCRATCH
    +----------+----------+
>34 !         PDTR9         !   R9  - DSR SCRATCH
    +----------+----------+
>36 !         PDTR10        !   R10 - DSR SCRATCH
    +----------+----------+
>38 !         PDTR11        !   R11 - DSR SCRATCH
    +----------+----------+
>3A !         PDTCRU        !   R12 - CRU OR TILINE ADDRESS
    +----------+----------+
>3C !         PDTR13        !   R13 - SAVED WP
    +----------+----------+
>3E !         PDTR14        !   R14 - SAVED PC
    +----------+----------+
>40 !         PDTR15        !   R15 - SAVED ST
    +----------+----------+
>42 !  PDTERR  !  PDTRTY    !   SAVED ERROR CODE FOR SYS LOG
    +----------+----------+       RETRIES ATTEMPTED COUNT
>44 !         PDTRC         !   READ  REQUEST COUNT
    +----------+----------+
>46 !         PDTWC         !   WRITE REQUEST COUNT
    +----------+----------+
>48 !         PDTMC         !   MISC  REQUEST COUNT
    +----------+----------+
>4A !         PDTREC        !   READ  ERROR COUNT
    +----------+----------+
>4C !         PDTWEC        !   WRITE ERROR COUNT
    +----------+----------+
>4E !         PDTMEC        !   MISC  ERROR COUNT
    +----------+----------+
>50 !         PDTSL1        !   SYSTEM LOG INFO
    +----------+----------+
>52 !         PDTSL2        !   SYSTEM LOG INFO
    +----------+----------+
>54 !         PDTBLN        !   MAXIMUM BUFFER LENGTH
    +----------+----------+
>56 !         PDTTM1        !   TIME OUT COUNT
    +----------+----------+
>58 !         PDTTM2        !   TIME OUT COUNT DOWN
    +----------+----------+
>5A !         PDTHRQ        !   HIDDEN REQUEST QUEUE
    +----------+----------+
>5C !         PDTWQ         !   WAITING REQUEST QUEUE
    +----------+----------+
>5E !         PDTSRB        !   SAVED REQUEST ADDRESS
    +----------+----------+
>60 !         PDTERQ        !   END -OF-RECORD  QUEUE
    +----------+----------+
>62 !         PDTSRQ        !   SPENT REQUEST QUEUE
```

```
       +----------+----------+
   >64 !        PDTPDS       !   PRIORITY DSR SCHEDULER QUE LINK
       +----------+----------+


   FLAGS FOR FIELD: PDTFLG     #0E - DEVICE STATUS FLAGS EXTENSION

     DFGIRB = (X...............) - COPY IRB TO SYSTEM LOG
            = (.XX.............) - RESERVED
     PDFSTA = (...XX...........) - DEVICE STATE
*                                  00 ONLINE       01 OFFLINE
*                                  10 DIAGNOSTIC   11 SPOOLER
     DFGOPF = (.....X..........) - DEVICE OPERATION FAILED
            = (......X.........) - RESERVED
     DFGVRT = (.......X........) - VIRTUAL DEVICE FLAG
            = (........XXXXXXXX) - NIO-KEYBOARD BID OWNER TASK RUN ID


   FLAGS FOR FIELD: PDTDSF     #26 - R2  - DEVICE STATUS FLAGS

     DSFCMO = (X...............) - OPENED WITH COMM OPEN (>4E)
     DSFAID = (.X..............) - USE ALTERNATE PDT
     DSFBI  = (..X.............) - BUFFER INPUT  (1=YES)
     DSFBO  = (...X............) - BUFFER OUTPUT (1=YES)
     DSFJIS = (....X...........) - JISCII FLAG(KATAKANA)
     DSFREN = (.....X..........) - RE-ENTER-ME
     DSFJAR = (......X.........) - JISCII RECEIVE  MODE
     DSFJAT = (.......X........) - JISCII TRANSMIT MODE
            = (........XX......) - RESERVED
     DSFWPM = (..........X.....) - WORD PROCESSING MODE
     DSFIRE = (...........X....) - INITIAL REQUEST ENTRY
     DSFINT = (............XXXX) - DEVICE INTERRUPT LEVEL MASK


   FLAGS FOR FIELD: PDTDTF     #28 - R3  - DEVICE TYPE FLAGS

     DTFFIL = (X...............) - FILE ORIENTED
     DTFTIL = (.X..............) - TILINE DEVICE
     DTFTIM = (..X.............) - ENABLE TIME-OUT
     DTFPRI = (...X............) - PRIVILEDGED DEVICE
     DTFKSB = (....X...........) - TERMINAL WITH A KSB
     DTFCOM = (.....X..........) - COMM DEVICE
     DTFSYD = (......X.........) - SYSTEM DISC
            = (.......X........) - RESERVED
```

EQUATES:

LABEL	EQUATE TO	VALUE	DESCRIPTION
PDFDSM	>1800	>1800	DEVICE STATE MASK
PDTOCN	PDTMAP	>10	VIRT PDTS ONLY-OWNER CHANNEL LEN/NA
PDTRDN	PDTR5	>2C	VIRT PDTS ONLY-REMOTE DEVICE LEN/NA

```
    PDTRDJ   PDTCRU          >3A   VIRT PDTS ONLY-REMOTE DEVICE JOB ID
    PDTSIZ   $               >66
```

```
************************************************************
*                                                          *
* PROGRAM FILE DIRECTORY INDEX ENTRY   (PFI)     04/20/79  *
*                                                          *
*            LOCATION: DISK                                *
************************************************************
* THE PFI IS USED TO DESCRIBE AN ENTRY IN A PROGRAM FILE.
* ENTRIES CAN BE TASK SEGMENTS, PROCEDURE SEGMENTS, PROGRAM
* SEGMENTS, AND OVERLAYS.  IN ADDITION TO A COMMON FIRST
* PORTION, THERE IS A SEPARATE VARIANT FOR EACH TYPE OF
* ENTRY. IN THE FLAG COMMENTS, T INDICATES THE COMMENT APPLIES
* TO A TASK ENTRY, P TO A PROCEDURE ENTRY, S TO A PROGRAM
* SEGMENT ENTRY AND O TO AN OVERLAY ENTRY.


      *----------+----------*
  >00 !        PFILEN       !   SEGMENT LENGTH (BYTES)
      +----------+----------+
  >02 !        PFIFLG       !   FLAGS
      +----------+----------+
  >04 !        PFIREC       !   RECORD NUMBER OF START OF IMAGE
      +----------+----------+
  >06 !        PFIDAT       !   DATE INSTALLED IN JULIAN FORMAT
      +----------+----------+
  >08 !        PFILOD       !   LOAD ADDRESS IN TASK
      +----------+----------+

                            TYPE DEPENDENT DATA (ANY SET)
      *----------+----------*
  >0A !  PFIVAR  !          !   SINGLE PORTION OF DATA
      +----------+----------+
  >0C !          !          !
      +----------+----------+
  >0E !          !          !
      +----------+----------+

                            TASK ENTRY DESCRIPTION
      *----------+----------*
  >0A !  PFIOVL  !  PFIPRI  !   OVERLAY LINK
      +----------+----------+      TASK PRIORITY
  >0C !  PFISG1  !  PFISG2  !   ID OF PROCEDURE 1 FOR TASK
      +----------+----------+      ID OF PROCEDURE 2 FOR TASK
  >0E !        PFITND       !   TASK LENGTH
      +----------+----------+

                            OVERLAY ENTRY DESCRIPTION
      *----------+----------*
  >0A !  PFIOV2  !  PFITID  !   OVERLAY LINK
      +----------+----------+      ID OF ASSOCIATED TASK
  >0C !        PFIOND       !   RESERVED (SET TO ZERO)
      +----------+----------+
                              PROCEDURE/PROGRAM ENTRY DESCRIPTION
     *----------+----------*
 >0A !       PFIPND        !   RESERVED (SET TO ZERO)
     +----------+----------+


 FLAGS FOR FIELD: PFIFLG     #02 - FLAGS

   PFFPRI = (X...............) - PRIVILEGED (T)
   PFFSYS = (.X..............) - SYSTEM (T,S)
   PFFRES = (..X.............) - MEMORY RESIDENT (T,P,S)
   PFFDEL = (...X............) - DELETE PROTECTED(T,P,S,O)
   PFFREP = (....X...........) - REPLICATABLE (T,S)
   PFFSG1 = (.....X..........) - PROC 1 IS ON THE PROG FILE (T)
   PFFSG2 = (......X.........) - PROC 2 IS ON THE PROG FILE (T)
   PFFUSE = (.......X........) - PFI ENTRY IS IN USE (T,P,S,O)
   PFFOVF = (........X.......) - OVERFLOW (T)
   PFFWCS = (.........X......) - WRITEABLE CONTROL STORE (T,P,S)
   PFFEXP = (..........X.....) - EXECUTE PROTECTED (T,P,S)
   PFFWRP = (...........X....) - WRITE PROTECTED (P,S)
   PFFUPD = (............X...) - UPDATABLE (T,S)
   PFFREU = (.............X..) - REUSABLE (T,S)
   PFFCPY = (..............X.) - COPYABLE (T,S)
   PFFSEC = (...............X) - SECURITY BYPASS (T)
*


 EQUATES:

    LABEL    EQUATE TO    VALUE   DESCRIPTION
    -----    ---------    -----   ------------------------------
    PFFRED    PFFPRI        >00    READABLE (P)
    PFFSHR    PFFSG1        >05    SEG. IS SHARE PROTECTED (S)
    PFFSPR    PFFWRP        >0B    SOFTWARE PRIVILEGED (T)
    PFISIZ    $             >10
```

```
************************************************************
*                                                          *
*      PROGRAM FILE RECORD ZERO        (PFZ)      3/7/78   *
*                                                          *
*           LOCATION: DISK                                 *
************************************************************
* THE PFZ DESCRIBES THE FIRST RECORD (RECORD 0) OF THE
* PROGRAM FILE.  IT INCLUDES BIT MAPS FOR ALL ELEMENTS IN
* THE PROGRAM FILE AS WELL AS DATA ABOUT CURRENT USE OF
* THE FILE.


      *----------+----------*
  >00 !  PFZRES  !          !   RESERVED
      +----------+----------+
      /          /          /
      /          /          /
      +----------+----------+
  >14 !  PFZMRT  !          !   BIT MAP - MEMORY-RESIDENT TASKS
      +----------+----------+
      /          /          /
      /          /          /
      +----------+----------+
  >34 !  PFZMRP  !          !   BIT MAP - MEMORY-RESIDENT PROCEDURES
      +----------+----------+
      /          /          /
      /          /          /
      +----------+----------+
  >54 !  PFZTSK  !          !   BIT MAP - ALL TASKS
      +----------+----------+
      /          /          /
      /          /          /
      +----------+----------+
  >74 !  PFZPRC  !          !   BIT MAP - ALL PROCEDURES
      +----------+----------+
      /          /          /
      /          /          /
      +----------+----------+
  >94 !  PFZNRT  !          !   BIT MAP - NONREPLICATABLE TASKS
      +----------+----------+
      /          /          /
      /          /          /
      +----------+----------+
  >B4 !  PFZOVL  !          !   BIT MAP - ALL OVERLAYS
      +----------+----------+
      /          /          /
      /          /          /
      +----------+----------+
  >D4 !  PFZMNT  !  PFZTO   !   MAXIMUM NUMBER OF TASKS
      +----------+----------+      FIRST TASK DIRECTORY ENTRY OFFSET
  >D6 !       PFZTR         !   FIRST TASK DIRECTORY ENTRY REC #
      +----------+----------+
>D8 !  PFZMNP   !  PFZPO   !     MAXIMUM NUMBER OF PROCEDURES
    +----------+----------+         FIRST PROC DIRECTORY ENTRY OFFSET
>DA !        PFZPR        !     FIRST PROC DIRECTORY ENTRY REC #
    +----------+----------+
>DC !  PFZMNO   !  PFZOO   !     MAXIMUM NUMBER OF OVERLAYS
    +----------+----------+         FIRST OVLY DIRECTORY ENTRY OFFSET
>DE !        PFZOR        !     FIRST OVLY DIRECTORY ENTRY REC #
    +----------+----------+
>E0 !        PFZMNH       !     MAXIMUM NUMBER OF HOLES
    +----------+----------+
>E2 !        PFZHO        !     FIRST AVAILABLE SPACE LIST OFFSET
    +----------+----------+
>E4 !        PFZHR        !     FIRST AVAILABLE SPACE LIST REC #
    +----------+----------+
```

EQUATES:

LABEL	EQUATE TO	VALUE	DESCRIPTION
PFZSIZ	$	>E6	SIZE OF INFORMATION SECTION OF REC

```
***********************************************************
*                                                         *
*  QUEUE HEADER              (QHR)            09/06/79    *
*                                                         *
*           LOCATION: SYSTEM ROOT, JCA                    *
***********************************************************
* QUEUE HEADERS FOR SYSTEM QUEUE SERVERS THAT RUN IN THE
* SYSTEM JOB ARE BUILT DURING SYSTEM GENERATION IN THE
* SYSTEM ROOT.  QUEUE HEADERS FOR SYSTEM QUEUE SERVERS
* THAT RUN IN A USER'S JOB ARE BUILD IN THE JCA WHEN THE
* JOB IS CREATED.  EACH QUEUE HEADER FOLLOWS THE QHR FORM.


      *----------+----------*
  >00 !        QHRNEW       !   ADDRESS OF NEWEST ENTRY
      +----------+----------+
  >02 !        QHROLD       !   ADDRESS OF OLDEST ENTRY
      +----------+----------+
  >04 !  QHRCNT  !  QHRTID  !   NUMBER OF ENTRIES ON QUEUE
      +----------+----------+      SERVER TASK ID
  >06 !        QHRTSB       !   TSB ADDRESS OF SERVER TASK
      +----------+----------+
  >08 !        QHRJSB       !   JSB ADDRESS OF SERVER TASK
      +----------+----------+
  >0A !  QHRLUN  !  FILL00  !   PROGRAM FILE LUNO FOR SERVER
      +----------+----------+      RESERVED
```

EQUATES:

LABEL	EQUATE TO	VALUE	DESCRIPTION
QHRSIZ	$	>0C	

```
************************************************************
*                                                          *
*  QUEUED IPC REQUEST        (QIR)        8/15/81          *
*                                                          *
*         LOCATION: SYSTEM AREA                            *
************************************************************
*  THE QIR IS PUT ON THE IPC TASK QUEUE WHEN AN IPC REQUEST
*  CANNOT BE PROCESSED IN FAST TRANSFER IPC.
                          ** BEGINNING PACKED RECORD QIR


      *----------+----------*
  >00 !       QIRQIR        !   POINTER TO NEXT QIR
      +----------+----------+
  >02 !       QIRJSB        !   JSB ADDRESS OF OWNER
      +----------+----------+      (0 IF GLOBAL CHANNEL)
  >04 !       QIRCCB        !   ADDRESS OF CCB TO BE PROCESSED
      +----------+----------+
  >06 SIZE                ** END OF PACKED RECORD
```

```
*************************************************************
*                                                           *
*  REQUEST DESCRIPTION BLOCK  (RDB)             05/21/79   *
*                                                           *
*      LOCATION: RPSDAT AND SOME SVC PROCESSORS             *
*************************************************************
* THE RDB FOR A GIVEN SVC SPECIFIES HOW TO BUFFER THE USER'S
* REQUEST FOR PROCESSING BY THE SVC PROCESSOR.  THE RDB IS
* LOCATED IN THE MODULE RPSDAT BUILT DURING SYSTEM GENERATION
* IF THE SVC IS AN OPTIONAL SVC OR IF THE SVC IS PROCESSED BY
* A QUEUE SERVER TASK.  OTHERWISE, THE RDB IS LOCATED IN
* THE FIRST PROCESSOR MODULE FOR THE SVC PROCESSOR.  AN RDB
* EXISTS FOR A GIVEN SVC ONLY IF THE CALL BLOCK MUST BE
* BUFFERED INTO A PORTION OF MEMORY FOR THE DURATION OF THE
* PROCESSING.


      *----------+----------*
  >00 !        RDBFLG       !   DESCRIPTION FLAGS
      +----------+----------+
  >02 !        RDBSRV       !   ADDRESS OF PROCESSOR ENTRY OR
      +----------+----------+      QUEUE HDR ADDRESS(RDFQIJ=0) OR
  >04 !        RDBRIB       !   ADDRESS OF RETURN INFORMATION BLOCK
      +----------+----------+
  >06 !        RDBMAX       !   MAXIMUM BUFFER LENGTH (BYTES)
      +----------+----------+
  >08 !  RDBBAS  !  RDBACC  !   BASIC REQUEST BLOCK LENGTH (BYTES)
      +----------+----------+      ACCOUNTING WEIGHTING FACTOR
  >0A !        FILL01       !   RESERVED FOR FUTURE USE
      +----------+----------+
  >0C !  RDBEXP  !  RDBLEN  !   EXPANSION FLAGS
      +----------+----------+      EXPANSION LENGTH TO BUFFER
  >0E !  RDBCOF  !  RDBBOF  !   OFFSET IN CALL BLOCK
      +----------+----------+      OFFSET IN BRB (0=CONTINUE FROM LAST


  FLAGS FOR FIELD: RDBFLG    #00 - DESCRIPTION FLAGS

    RDFEXT = (X...............) - EXTENSIONS TO THE RDB(1=YES)
    RDFRST = (.X..............) - REQUIRES SYSTEM TASK (1=YES)
    RDFRPT = (..X.............) - REQUIRES SOFT.PRIVILEGED TASK
    RDFQOP = (...X............) - QUEUE SERVER(0) OR PROCESSOR
    RDFSOD = (....X...........) - STATIC(0) OR DYNAMIC BUFFER
    RDFDSJ = (.....X..........) - DYNAMIC BUFFER - STA(0) OR JCA
    RDFREV = (......X.........) - REVISING A BUFFER (1=YES)
    RDFINT = (.......X........) - CAN(1) OR CANNOT(0) BE
*                                 AN INITIATED EVENT
    RDFQIJ = (........X.......) - QUEUE HDR IN STA(0) OR JCA
           = (.........XXXXXXX) - RESERVED


  FLAGS FOR FIELD: RDBEXP    #0C - EXPANSION FLAGS
```

```
    RDFTYP = (X...............) - TYPE OF CALL BLOCK OFFSET PTR
*                                  0=START OF DATA WITH RDBLEN
*                                    BYTES TO BUFFER
*                                  1=POINTER TO EXPANSION
*                                    BLOCK WITH OWN LENGTH
*                                    BYTE AND DATA BUFFER
    RDFJCA = (.X..............) - 1=BUFFER THIS IN JCA BY ITSELF
    RDFMOR = (..X.............) - MORE EXPANSION BLOCKS (1=YES)
*                                 (AFTER THIS ONE)
    RDFJAV = (...X............) - 1=WERE ABLE TO GET JCA SPACE
           = (....XXXX........) - RESERVED
```

EQUATES:

LABEL	EQUATE TO	VALUE	DESCRIPTION
RDBSIZ	$	>10	

```
************************************************************
*                                                          *
*  RETURN INFORMATION BLOCK   (RIB)             02/08/79   *
*                                                          *
*            LOCATION: RPSDAT AND SVC PROCESSORS           *
************************************************************
* THE RIB FOR A GIVEN SVC TELLS HOW MUCH AND WHERE TO
* RETURN INFORMATION FROM A BUFFERED CALL BLOCK THAT WAS
* USED BY THE SVC PROCESSOR.  THE RIB IS IN THE MODULE
* RPSDAT BUILT DURING SYSTEM GENERATION IF THE SVC IS AN
* OPTIONAL SVC OR IS ONE PROCESSED BY A QUEUE SERVER TASK.
* OTHERWISE THE RIB IS IN ONE OF THE SVC PROCESSOR MODULES.
* THE RIB FOR A PARTICULAR SVC IS ACTUALLY SPECIFIED AS ONE
* FIELD FOR RIBPRO, THEN ANY NUMBER OF PAIRS OF VALUES FOR
* OFFSET AND LENGTH, THEN A WORD OF ZERO TO TERMINATE THE RIB.


    *----------+----------*
>00 !       RIBPRO        !   POSTPROCESSOR (IF SPECIAL ONE)
    +----------+----------+
>02 !  RIBOFF  !  RIBLEN  !   CALL BLOCK OFFSET
    +----------+----------+      LENGTH TO UNBUFFER (BYTES)


EQUATES:

   LABEL   EQUATE TO   VALUE   DESCRIPTION
   -----   ---------   -----   -------------------------------
   RIBSIZ  $             >04
```

```
***************************************************************
*  RECORD LOCK TABLE          (RLT)       05/09/79            *
*                                                             *
*      LOCATION: SYSTEM TABLE AREA OR USER JCA                *
*                         (WHEREVER FCB IS LOCATED)           *
***************************************************************
* FOR A FILE WHICH HAS LOCKED RECORDS, EACH LOCKED RECORD IS
* REPRESENTED BY A RLT CHAINED TO THE FILE CONTROL BLOCK OF
* THAT FILE.

      *----------+----------*
  >00 !       RLTRLT        !   NEXT TABLE ENTRY ADDRESS
      +----------+----------+
  >02 !       RLTLDT        !   LOCKING LDT ADDRESS
      +----------+----------+
  >04 !       RLTTSB        !   LOCKING TSB ADDRESS
      +----------+----------+
  >06 !       RLTJSB        !   OWNER JSB ADDRESS
      +----------+----------+
  >08 !       RLTBN         !   LOCKED BLOCK NUMBER
      +----------+----------+
  >0A !                     !
      +----------+----------+
  >0C !       RLTOFF        !   LOCKED OFFSET
      +----------+----------+

  EQUATES:

     LABEL    EQUATE TO    VALUE    DESCRIPTION
     -----    ---------    -----    ------------------------------
     RLTSIZ   $               >0E
```

```
***********************************************************
*                                                         *
*  RESOURCE OWNERSHIP BLOCK  (ROB)              08/29/83  *
*                                                         *
*            LOCATION: JCA                                *
***********************************************************
* AN ROB IS BUILT FOR AN I/O RESOURCE WHEN AN ATTACH RESOURCE
* OPERATION IS PERFORMED.  THE ROB IS LINKED INTO THE ROB
* LIST ANCHORED IN THE JCA.


      *----------+----------*
  >00 !        ROBROB       !   NEXT ROB ADDRESS
      +----------+----------+
  >02 !  ROBACT  !  ROBATN  !   ATTACHED COUNT
      +----------+----------+      ATTACH NUMBER
  >04 !        ROBFMT       !
      +----------+----------+
  >06 !        ROBFCB       !
      +----------+----------+

  EQUATES:

     LABEL   EQUATE TO   VALUE   DESCRIPTION
     -----   ---------   -----   ------------------------------
     ROBSIZ  $              >08
```

```
************************************************************
*                                                          *
*  RESOURCE PRIVILEGE BLOCK   (RPB)               08/30/83 *
*                                                          *
*           LOCATION: SYSTEM AREA OR JCA                   *
************************************************************
* AN RPB IS BUILT FOR AN I/O RESOURCE WHEN A LUNO IS ASSIGNED.
* IT IS ATTACHED TO THE APPROPRIATE RESOURCE STRUCTURE: CCB,
* FCB, OR PDT.


      *----------+----------*
  >00 !       RPBRPB        !   LINK TO NEXT RPB
      +----------+----------+
  >02 !  RPBFLG  !  RPBCFI  !   FLAG BYTE
      +----------+----------+      CURRENT FILE INDEX (CONCAT. FILES)
  >04 !       RPBLDT        !   LDT ADDRESS
      +----------+----------+
  >06 !       RPBJSB        !   JSB ADDRESS
      +----------+----------+
  >08 !       RPBLRN        !   LOGICAL RECORD NUMBER
      +----------+----------+
  >0A !                     !
      +----------+----------+
  >0C !       RPBBN         !   BLOCK NUMBER
      +----------+----------+
  >0E !                     !
      +----------+----------+
  >10 !       RPBOCB        !   OFFSET IN CURRENT BLOCK
      +----------+----------+


  FLAGS FOR FIELD: RPBFLG    #02 - FLAG BYTE

    RPFATT = (X...............) - 1 = ATTACHED
    RPFOPN = (.X..............) - 1 = LUNO OPEN
    RPFFBS = (..X.............) - 1 = FORWARD OR BACK SPACE
           = (...XXX..........) - RESERVED
    RPFACU = (......XX........) - ACCESS PRIVILEGES IN USE
*                                 00 = EXCLUSIVE WRITE
*                                 01 = EXCLUSIVE ALL
*                                 10 = SHARED
*                                 11 = READ ONLY


  EQUATES:

     LABEL    EQUATE TO   VALUE   DESCRIPTION
     -----    ---------   -----   ------------------------------
     RPBACM   >0300          >300   ACCESS PRIVILEGES BIT MASK
     RPBMSZ   $              >08   MINIMUM SIZE
     RPBSIZ   $              >12   RPB SIZE
```

```
************************************************************
*                                                          *
*      RESERVE SEGMENT TABLE          (RST)   09/21/81     *
*                                                          *
*           LOCATION:  JCA                                 *
************************************************************
* THE RST DESCRIBES ALL OF THE SEGMENTS THAT A JOB HAS
* RESERVED WITH A RESERVE SEGMENT SVC CALL.
* EACH ENTRY IS:
* SEGMENT SSB ADDRESS
* SEGMENT SMT ADDRESS

                              MAX NUMBER OF ENTRIES IN RST
     *----------+----------*
 >00 !        RSTRST       !  LINK TO NEXT RST
     +----------+----------+
 >02 !        RSTSID       !  ID'S OF RESERVED SEGMENTS (MAX 8)
     +----------+----------+     (ZERO IF ENTRY NOT IN USE)
     /          /          /
     /          /          /
     +----------+----------+
 >22 !  FILL00  !  RSTALC  !  RESERVED FOR FUTURE USE
     +----------+----------+     NUMBER OF ALLOCATED ENTRIES (MAX 8)
```

EQUATES:

LABEL	EQUATE TO	VALUE	DESCRIPTION
RSTCNT	8	>08	MAX NUMBER OF ENTRIES IN RST
RSTSIZ	$	>24	

```
************************************************************
*                                                          *
*  SECONDARY ALLOCATION TABLE (SAT)              01/25/77  *
*                                                          *
*            LOCATION:JCA OR SYSTEM AREA, WITH FCB         *
************************************************************
* THE SAT SHOWS THE NUMBER AND LOCATION OF SECONDARY FILE
* ALLOCATIONS.

                                  COUNT OF SAT ENTRIES
      *----------+----------*
  >00 !        SATASZ       !    ALLOCATION SIZE
      +----------+----------+
  >02 !        SATADU       !    ALLOCATION START
      +----------+----------+
  >04 !  FILL00  !          !    REMAINDER OF BLOCK
      +----------+----------+
      /          /          /
      /          /          /
      +----------+----------+

  EQUATES:

     LABEL    EQUATE TO    VALUE    DESCRIPTION
     -----    ---------    -----    ------------------------------
     SATNSA    16            >10      COUNT OF SAT ENTRIES
     SATSIZ    $             >40
```

```
***********************************************************
*                                                         *
*   TRACK 0, SECTOR 0     (SCO)                  09/09/83 *
*                                                         *
*             LOCATION: TRACK 0, SECTOR 0 OF EACH DISK    *
***********************************************************


     *----------+----------*
 >00 !  SCOVNM  !          !   VOLUME NAME
     +----------+----------+
     /          /          /
     /          /          /
     +----------+----------+
 >08 !       SCOTNA        !   TOTAL SCO ADU ON DISK
     +----------+----------+
 >0A !  SCOSBM  !  SCOTBM  !   STARTING SECTOR OF BIT MAPS
     +----------+----------+      TOTAL SCO BIT MAPS
 >0C !       SCORL         !   TRACK 0 RECORD LENGTH
     +----------+----------+
 >0E !       SCOSLT        !   SYSTEM LOADER TRACK ADDRESS
     +----------+----------+
 >10 !  FILL00  !          !   * * RESERVED * *
     +----------+----------+
 >12 !          !          !
     +----------+----------+
 >14 !          !          !
     +----------+----------+
 >16 !       SCONBA        !   TOTAL SCO BAD ADU ON DISK
     +----------+----------+
 >18 !       SCOSLE        !   SYSTEM LOADER ENTRY POINT
     +----------+----------+
 >1A !       SCOSLL        !   SYSTEM LOADER LENGTH
     +----------+----------+
 >1C !  FILL01  !          !   * * RESERVED * *
     +----------+----------+
     /          /          /
     /          /          /
     +----------+----------+
 >24 !       SCOLT1        !   SYSTEM LOADER TRACK (COPY 2)
     +----------+----------+
 >26 !  FILL02  !          !   * * RESERVED * *
     +----------+----------+
     /          /          /
     /          /          /
     +----------+----------+
 >2E !  SCOPI1  !          !   PRIMARY SYSTEM FILE NAME
     +----------+----------+
     /          /          /
     /          /          /
     +----------+----------+
 >36 !  SCOPI2  !          !   SECONDARY SYSTEM FILE NAME
```

```
    +----------+----------+
    /          /          /
    /          /          /
    +----------+----------+
>3E !        SCOPIF       !   SYSTEM SELECTOR
    +----------+----------+
>40 !        SCOVDA       !   VOLUME DIRECTORY ADU SCO
    +----------+----------+
>42 !        SCOVPL       !   VCATALOG PHYSICAL RECORD LENGTH
    +----------+----------+
>44 !        SCOSPA       !   SECTORS/ADU
    +----------+----------+
>46 !  SCODCD  !          !   DISK CREATION DATE
    +----------+----------+
>48 !          !          !
    +----------+----------+
>4A !  SCOPF1  !          !   PRIMARY PROGRAM FILE
    +----------+----------+
    /          /          /
    /          /          /
    +----------+----------+
>52 !  SCOPF2  !          !   SECONDARY PROGRAM FILE
    +----------+----------+
    /          /          /
    /          /          /
    +----------+----------+
>5A !        SCOPFF       !   PROGRAM FILE SWITCH
    +----------+----------+
>5C !  SCOOF1  !          !   PRIMARY OVERLAY FILE
    +----------+----------+
    /          /          /
    /          /          /
    +----------+----------+
>64 !  SCOOF2  !          !   SECONDARY OVERLAY FILE
    +----------+----------+
    /          /          /
    /          /          /
    +----------+----------+
>6C !        SCOOFF       !   OVERLAY FILE SWITCH
    +----------+----------+
>6E !  SCOIL1  !          !   PRIMARY INTERMEDIATE LOADER
    +----------+----------+
    /          /          /
    /          /          /
    +----------+----------+
>76 !  SCOIL2  !          !   SECONDARY INTERMEDIATE LOADER
    +----------+----------+
    /          /          /
    /          /          /
    +----------+----------+
>7E !        SCOILF       !   INTERMEDIATE LOADER FLAG
    +----------+----------+
>80 !  SCODIN  !          !   DIAGNOSTIC FILE NAME
```

```
     +----------+----------+
     /          /          /
     /          /          /
     +----------+----------+
>88  !        SCODIF       !   DIAGNOSTIC FLAG
     +----------+----------+
>8A  !        SCODRS       !   DBUILD DETERMINES DEFAULT PRS
     +----------+----------+
>8C  !        SCOBAL       !   STARTING SECTOR OF BAD ADU LIST
     +----------+----------+
>8E  !        SCOSPR       !   TRACK 0 SECTORS PER RECORD
     +----------+----------+
>90  !   SCOWF1  !         !   WCS PRIMARY MICROCODE FILE
     +----------+----------+
     /          /          /
     /          /          /
     +----------+----------+
>98  !   SCOWF2  !         !   WCS SECONDARY MICROCODE FILE
     +----------+----------+
     /          /          /
     /          /          /
     +----------+----------+
>A0  !        SCOWFF       !   WCS FLAG SWITCH
     +----------+----------+
>A2  !        SCOVIF       !   TRACK 1 SELECT FLAG
     +----------+----------+
>A4  !        SCOSTA       !   STATE OF DISK
     +----------+----------+      1 - NOT ANALYZED
>A6  !        SCODCT       !   DISK CREATION TIME
     +----------+----------+
>A8  !        SCOFSF       !   * * RESERVED * *
     +----------+----------+
```

EQUATES:

LABEL	EQUATE TO	VALUE	DESCRIPTION
SCOSIZ	$	>AA	

```
***********************************************************
*                                                         *
*    STAGE DESCRIPTOR BLOCK  (SDB)                07/16/81 *
*                                                         *
*    LOCATION:  A NAME DEFINITION SEGMENT                 *
***********************************************************


      *----------+----------*
  >00 !        SDBSDB        !   FIXED LINK
      +----------+----------+
  >02 !  SDBCID  !  SDBSNO   !   CREATOR TASK ID
      +----------+----------+      STAGE NUMBER
  >04 !  SDBTCT  !  SDBRES   !   TASK COUNT
      +----------+----------+      RESERVED
  >06 !        SDBPAR        !   POINTER TO PARENT SDB
      +----------+----------+
  >08 !        SDBDEL        !   DESCENDANT ERROR LIST ANCHOR
      +----------+----------+
************************************************************
*                                                          *
*  SEGMENT GROUP BLOCK   (SGB)                  04/09/81   *
*                                                          *
*            LOCATION: SEGMENT MANAGER TABLE AREA          *
************************************************************
* THE SGB IS AN ANCHOR FOR SSBS OF SEGMENTS WHICH FORM A
* LOGICAL SET.  IT IS USED TO ACCESS SSBS FOR SEGMENT
* MANAGER CALLS MADE BY LUNO.


      *----------+----------*
  >00 !       SGBSGB        !   POINTER TO NEXT SGB IN TABLE
      +----------+----------+
  >02 !       SGBOMT        !   SMT SSB POINTER FOR OVERFLOW SGB
      +----------+----------+
  >04 !       SGBOGB        !   SGB SSB POINTER FOR OVERFLOW SGB
      +----------+----------+
  >06 !       SGBSSB        !   SSB LIST HEADER
      +----------+----------+
  >08 !       SGBFLG        !   FLAGS
      +----------+----------+
  >0A !       SGBFMT        !   FDP FOR THE SEGMENT GROUP
      +----------+----------+
  >0C !       SGBFCB        !
      +----------+----------+


  FLAGS FOR FIELD: SGBFLG    #08 - FLAGS

    SGFPPL = (X...............) - PROGRAM FILE SEGMENT GROUP
    SGFDFL = (.X..............) - DATA FILE SEGMENT GROUP
    SGFMBS = (..X.............) - MEMORY-BASED SEGMENT GROUP
    SGFRES = (...XXXXXXXXXXXXX) - RESERVED


  EQUATES:

     LABEL    EQUATE TO    VALUE   DESCRIPTION
     -----    ---------    -----   ------------------------------
     SGBOSB    SGBOMT        >02
     SGBOLK    SGBOGB        >04
     SGBSIZ    $             >0E
```

```
***********************************************************
*                                                         *
*   SYSTEM LOG BLOCK FORMATS   (SLB)           02/08/82 *
*                                                         *
*           LOCATION: SYSTEM AREA                         *
***********************************************************
*  THIS TEMPLATE INCLUDES FORMATS FOR SEVERAL TYPES OF SYSTEM
*  LOG MESSAGES.  EACH FORMAT INCLUDES THE SAME QUEUE LINK
*  FIELD AND FLAGS FIELD.  EACH ALSO HAS A 4 BYTE TYPE FIELD.
*  OTHER FIELDS ARE PARTICULAR TO A TYPE OF LOG BLOCK BEING
*  BUILT.

                           COMMON PORTION FOR ALL TYPES
     *----------+----------*
 >00 !       SLBSLB        !   QUEUE LINK
     +----------+----------+
 >02 !  SLBFLG  !  SLBCNT  !   BLOCK TYPE
     +----------+----------+      COUNT OF LOST MESSAGES
 >04 !       SLBDAY        !   BINARY DAY
     +----------+----------+
 >06 !       SLBHR         !   BINARY HOUR
     +----------+----------+
 >08 !       SLBMIN        !   BINARY MINUTES
     +----------+----------+
 >0A !  SLBTYP  !          !   LOG BLOCK TYPE
     +----------+----------+
     /          /          /
     /          /          /
     +----------+----------+

                           TYPE 1 - DEVICE ERROR WITH IMAGE
     *----------+----------*
 >12 !  SLBEC   !  SLBSTI  !   ERROR CODE
     +----------+----------+      STATION ID
 >14 !       SLBJOB        !   JOB ID
     +----------+----------+
 >16 !  SLBIID  !  SLBRID  !   TASK INSTALLED ID
     +----------+----------+      TASK RUN ID


     *----------+----------*
 >18 !  SLBLUN  !  SLBRTY  !   LUNO
     +----------+----------+      RETRY COUNT
 >1A !  SLBRSF  !  SLBACT  !   RETRY SUCCESS(0)/FAILURE(1)
     +----------+----------+      IMAGE WORD COUNT


     *----------+----------*
 >1C !       SLBAIM        !   AFTER IMAGE
     +----------+----------+
     /          /          /
     /          /          /
```

```
      +----------+----------+
  >2C !       SLBBIM        !   BEFORE IMAGE
      +----------+----------+
      /          /          /
      /          /          /
      +----------+----------+

                            TYPE 2 - DEVICE ERROR WITH CALL BLOC
      *----------+----------*
  >1C !       SLBIRB        !   SPACE FOR 12 BYTES OF CALL BLK
      +----------+----------+
      /          /          /
      /          /          /
      +----------+----------+

                            TYPE 3 - ABNORMAL TASK TERMINATION
      *----------+----------*
  >18 !       SLBWP         !   WORKSPACE POINTER AT ERROR
      +----------+----------+
  >1A !       SLBPC         !   PROGRAM COUNTER
      +----------+----------+
  >1C !       SLBST         !   STATUS AT ERROR
      +----------+----------+

                            TYPE 4 - STATISTICS FROM A DSR
      *----------+----------*
  >12 !       SLBRDG        !   NUMBER OF GOOD READS
      +----------+----------+
  >14 !       SLBWRG        !   NUMBER OF GOOD WRITES
      +----------+----------+
  >16 !       SLBOTG        !   NUMBER OF GOOD OTHER OPS
      +----------+----------+
  >18 !       SLBRDB        !   NUMBER OF BAD READS
      +----------+----------+
  >1A !       SLBWRB        !   NUMBER OF BAD WRITES
      +----------+----------+
  >1C !       SLBOTB        !   NUMBER OF BAD OTHER OPS
      +----------+----------+

                            TYPE 5 - USER ISSUED SYSTEM LOG SVC
      *----------+----------*
  >12 !  SLBLEN  !  FILL00  !   LENGTH OF USER MESSAGE
      +----------+----------+      RESERVED
  >14 !  SLBUMS  !          !   USER MESSAGE BEGINS HERE
      +----------+----------+
      /          /          /
      /          /          /
      +----------+----------+
>0112 !          !
      +----------+

                            TYPE 6 - MEMORY CACHE ERRORS
      *----------+----------*
```

```
>12 !  SLBANK  !  SLBPRA  !   BANK (A OR B)
    +----------+----------+      ADDRESS PARITY IN BANK A (G/B)
>14 !  SLBPRB  !  SLBBA6  !   ADDRESS PARITY IN BANK B (G/B)
    +----------+----------+      BASE ADDRESS OF CONTROLLER
>16 !  SLBME6  !  SLBEVN  !   AMOUNT OF MEMORY
    +----------+----------+      ERROR ON EVEN WORD (Y/N)
>18 !       SLBAD6        !   TPCS ADDRESS
    +----------+----------+

                         TYPE 7 - MEMORY PARITY ERRORS
    *----------+----------*
>12 !  SLBBIT  !  SLBROW  !   BIT IN ERROR
    +----------+----------+      ROW IN ERROR
>14 !  SLBCOR  !  SLBBA7  !   CORRECTABLE? (Y/N)
    +----------+----------+      BASE ADDRESS OF CONTROLLER
>16 !  SLBME7  !  SLBCTY  !   AMOUNT OF MEMORY
    +----------+----------+      CONTROLLER TYPE
>18 !       SLBAD7        !   TPCS ADDRESS
    +----------+----------+
```

EQUATES:

LABEL	EQUATE TO	VALUE	DESCRIPTION
SLBVR1	$	>12	
SLBVR2	$	>18	
SLBVR3	$	>1C	
SLBSZ1	$-SLBSLB	>3C	DEVICE MESSAGE SIZE
SLBSZ2	$-SLBSLB	>28	
SLBSZ3	$-SLBSLB	>1E	
SLBSZ4	$-SLBSLB	>1E	
SLBMXL	255	>FF	MAX USER LENGTH
SLBSZ5	$-SLBSLB	>113	
SLBSZ6	$-SLBSLB	>1A	
SLBSZ7	$-SLBSLB	>1A	

```
***********************************************************
*                                                         *
*  SEMAPHORE LIST HEADER      (SLH)            03/15/79   *
*                                                         *
*        LOCATION: JCA                                    *
***********************************************************
* THE SLH IS USED TO DESCRIBE SEMAPHORES USED IN JOBS.  FOR
* EACH SEMAPHORE IN USE, THERE IS A LIST HEADER SHOWING THE
* NUMBER OF THE SEMAPHORE, ITS VALUE, AND THE ENTRIES WAITING
* FOR SEMAPHORE ACTION.


      *----------+----------*
  >00 !        SLHSLH       !   NEXT SEMAPHORE ENTRY
      +----------+----------+
  >02 ! SLHVAL   !  SLHNUM  !   SEMAPHORE VALUE
      +----------+----------+      SEMAPHORE NUMBER
  >04 !        SLHNEW       !   ADDRESS OF NEWEST ENTRY
      +----------+----------+
  >06 !        SLHOLD       !   ADDRESS OF OLDEST ENTRY
      +----------+----------+
  >08 ! SLHCNT   !  SLHTID  !   NUMBER OF ENTRIES ON QUEUE
      +----------+----------+      SERVER TASK ID   (NOT USED)
  >0A !        SLHTSB       !   TSB ADDRESS OF SERVER TASK(NU)
      +----------+----------+


  EQUATES:

     LABEL   EQUATE TO   VALUE   DESCRIPTION
     -----   ---------   -----   ------------------------------
     SLHSIZ   $             >0C
```

```
**********************************************************
*  SEGMENT MANAGEMENT REQUEST   (SMR)        11/05/81      *
*                                                         *
*         LOCATION: SYSTEM TABLE AREA                     *
**********************************************************
* THE SMR IS A SEGMENT MANAGEMENT SVC BLOCK WITH SEVERAL
* ADDITIONAL FIELDS DEFINED FOR USE BY SEGMENT MANAGER
* DURING PROCESSING OF THE SVC.


      *----------+----------*
  >00 !  SMRSVC  !  SMRERR  !    SVC CODE
      +----------+----------+       ERROR CODE
  >02 !  SMROP   !  SMRLUN  !    SEGMENT MANAGER SUB-OPCODE
      +----------+----------+       LOGICAL UNIT
  >04 !       SMRFLG        !    FLAGS
      +----------+----------+
  >06 !       SMRNS1        !    NEW SEGMENT ID WORD 1
      +----------+----------+
  >08 !       SMRNS2        !    NEW SEGMENT ID WORD 2
      +----------+----------+
  >0A !       SMROSG        !    OLD SEGMENT ID
      +----------+----------+
  >0C !       SMRADR        !    SEGMENT ADDRESS
      +----------+----------+
  >0E !       SMRLEN        !    SEGMENT LENGTH
      +----------+----------+
  >10 !       SMRATR        !    SEGMENT ATTRIBUTE (AS IN SSB)
      +----------+----------+
  >12 !       SMRFMT        !    FDP ADDRESS
      +----------+----------+
  >14 !       SMRFCB        !
      +----------+----------+


  FLAGS FOR FIELD: SMRFLG    #04 - FLAGS

    SMFINS = (X...............) - INSTALLED ID
    SMFNMD = (.X..............) - NOT MODIFIED
    SMFREL = (..X.............) - RELEASABLE
    SMFMBS = (...X............) - MEMORY BASED
    SMFPOS = (....X...........) - 0 IF POSITION NUMBER SPECIFIED
*                                  FOR OLD SEGMENT. 1 IF RUNTIME
*                                  ID SPECIFIED.
    SMFTSK = (.....X..........) - TASK SEGMENT
    SMFVLD = (......X.........) - VERIFY PROG. FILE LOAD ADDR
    SMFSRE = (.......X........) - SET/RESET FLAG ENABLE
    SMFSEU = (........X.......) - SET/RESET FLAG
*                                  1=>SET EXCLUSIVE USE
*                                  0=>RESET EXCLUSIVE USE
    SMFSYS = (.........X......) - SYSTEM TASK
           = (..........XXXX..) - ***RESERVED***
```

```
SMFPSN = (..............XX) - POSITION NUMBER(1,2, OR 3)
```

EQUATES:

LABEL	EQUATE TO	VALUE	DESCRIPTION
SMRSIZ	$	>16	

```
*************************************************************
*                                                           *
*       SEGMENT MANAGER TABLE  (SMT)              1/30/79   *
*                                                           *
*       LOCATION: USER MEMORY                               *
*************************************************************
* THE SMT IS THE TEMPLATE FOR THE STATIC DEFINITIONS IN
* THE SEGMENT MANAGER SPECIAL TABLE AREAS.

                                    STARTS AFTER MM OVERHEAD
      *----------+----------*
  >00 !        SMTSGB       !   SGB LIST HEADER
      +----------+----------+
  >02 !        SMTRID       !   LAST RUN ID ALLOCATED
      +----------+----------+
  >04 !  SMTMAP  !          !   ALLOCATED RUN ID BIT MAP
      +----------+----------+
      /          /          /
      /          /          /
      +----------+----------+
************************************************************
*                                                          *
*  SEGMENT OWNER BLOCK         (SOB)             09/23/81  *
*                                                          *
*           LOCATION: SMT                                  *
************************************************************
* THE SOB IS USED TO IDENTIFY THE TASK WHICH HAS EXCLUSIVE
* USE OF A SEGMENT.  IT IS LINKED TO THE SSB.

      *----------+----------*
  >00 !       SOBJSB        !   JSB ADDRESS OF SSB OWNER
      +----------+----------+
  >02 !       SOBTSB        !   TSB ADDRESS OF SSB OWNER
      +----------+----------+

  EQUATES:

     LABEL   EQUATE TO   VALUE   DESCRIPTION
     -----   ---------   -----   ------------------------------
     SOBSIZ   $             >04
```

```
*************************************************************
*                                                           *
*   SYSTEM OVERLAY LOAD TABLE   (SOV)             09/09/83 *
*                                                           *
*            LOCATION: SYSTEM ROOT                          *
*************************************************************
* THE SOV IS BUILT DURING SYSTEM GENERATION AS PART OF THE
* MODULE SOVT, DEPENDING ON THE OPTIONS CHOSEN DURING THE
* GENERATION.  THE FORMAT OF EACH ENTRY IS SHOWN IN THE OVT.
                        ** BEGINNING PACKED RECORD SOV


      *----------+----------*
  >00 !        FMORWT       !   FM REWRITE RECORD OVERLAY
      +----------+----------+
  >02 !                     !
      +----------+----------+
  >04 !                     !
      +----------+----------+
  >06 !        FMOMSC       !   FM WEOF,CLOSE/EOF,OPEN REWIND,UNLK
      +----------+----------+
  >08 !                     !
      +----------+----------+
  >0A !                     !
      +----------+----------+
  >0C !        FMOEXT       !   FM EXTEND FILE ALLOCATION OVERLAY
      +----------+----------+
  >0E !                     !
      +----------+----------+
  >10 !                     !
      +----------+----------+
  >12 !        FMOOPX       !   FM OPEN EXTEND OVERLAY
      +----------+----------+
  >14 !                     !
      +----------+----------+
  >16 !                     !
      +----------+----------+
  >18 !        KMOINS       !   KM INSERT PROCESSOR
      +----------+----------+
  >1A !                     !
      +----------+----------+
  >1C !                     !
      +----------+----------+
  >1E !        KMODLS       !   KM DELETE AND SET CURRENCY
      +----------+----------+
  >20 !                     !
      +----------+----------+
  >22 !                     !
      +----------+----------+
  >24 !        KMOOPC       !   KM OPEN AND CLOSE PROCESSORS
      +----------+----------+
  >26 !                     !
```

```
    +----------+----------+
>28 !                     !
    +----------+----------+
>2A !       KMOBDE        !   KM DELETE SUBROUTINES
    +----------+----------+
>2C !                     !
    +----------+----------+
>2E !                     !
    +----------+----------+
>30 !       KMOBTI        !   KM B-TREE SPLIT FOR KMBTI
    +----------+----------+
>32 !                     !
    +----------+----------+
>34 !                     !
    +----------+----------+
>36 !       KMORWS        !   KM REWRITE SUBROUTINES
    +----------+----------+
>38 !                     !
    +----------+----------+
>3A !                     !
    +----------+----------+
>3C !       DMALLC        !   DM ALLOCATION SCAN OVERLAY
    +----------+----------+
>3E !                     !
    +----------+----------+
>40 !                     !
    +----------+----------+
>42 !       DMCHPM        !   DM CHANGE PARTIAL BIT MAPS
    +----------+----------+
>44 !                     !
    +----------+----------+
>46 !                     !
    +----------+----------+
>48 !       CFDFOV        !   IU CREATE/DELETE FILE OVERLAY
    +----------+----------+
>4A !                     !
    +----------+----------+
>4C !                     !
    +----------+----------+
>4E !       OTHOV1        !   IU OTHER FUNCTIONS OVERLAY
    +----------+----------+
>50 !                     !
    +----------+----------+
>52 !                     !
    +----------+----------+
>54 !       OTHOV2        !   IU RF, AA, DA, CIC, DIC OVERLAY
    +----------+----------+
>56 !                     !
    +----------+----------+
>58 !                     !
    +----------+----------+
>5A !       PMERRS        !   PM ERROR PROCESSING SUBROUTINES
    +----------+----------+
>5C !                     !
    +----------+----------+
>5E !                     !
    +----------+----------+
>60 !       KMOPLR        !   KM PARTIAL LOG ERROR RECOVERY
    +----------+----------+
>62 !                     !
    +----------+----------+
>64 !                     !
    +----------+----------+
>66 !       SECMGR        !   IU SECURITY MANAGER OVERLAY
    +----------+----------+
>68 !                     !
    +----------+----------+
>6A !                     !
    +----------+----------+
>6C !       KMORWT        !   KM REWRITE MAIN ROUTINE
    +----------+----------+
>6E !                     !
    +----------+----------+
>70 !                     !
    +----------+----------+
>72 SIZE               ** END OF PACKED RECORD
```

```
************************************************************
*                                                          *
*      SEGMENT STATUS BLOCK (SSB)                09/09/83  *
*                                                          *
*           LOCATION:  ROOT AND SEGMENT MANAGER TABLE AREA *
************************************************************
* EACH SEGMENT WHICH IS IN MEMORY IS DESCRIBED BY AN SSB.
* THE SSB INCLUDES CHARACTERISTICS OF THE SEGMENT, LOCATION,
* AND USE INFORMATION.
*
* SPECIAL FIELD COMMENTS:
*
* SSBWCT -   THIS FIELD IS USED TO KEEP TRACK OF THE AMOUNT
*            OF FREE AREA IN A SPECIAL TABLE AREA.  APPLIES
*            TO SSB'S FOR SMT'S AND FMT'S.
*
* SSBID1/2 - THIS FIELD CONTAINS THE BLOCK NUMBER FOR A
*             SEGMENT WHICH IS ASSOCIATED WITH A DATA FILE.
*
```

```
     *----------+----------*
 >00 !       SSBSSB        !   SSB LINK
     +----------+----------+
 >02 !       SSBID1        !   SEGMENT INSTALLED ID FIRST WORD
     +----------+----------+
 >04 !       SSBID2        !   SEGMENT INSTALLED ID SECOND WORD
     +----------+----------+
 >06 !       SSBRID        !   SEGMENT RUN-TIME ID
     +----------+----------+
 >08 !       SSBATR        !   SEGMENT ATTRIBUTES
     +----------+----------+
 >0A !       SSBRCT        !   SEGMENT RESERVE COUNT
     +----------+----------+
 >0C !       SSBUCT        !   SEGMENT USE COUNT
     +----------+----------+
 >0E !       SSBSGB        !   SEGMENT GROUP BLOCK POINTER
     +----------+----------+
 >10 !       SSBADR        !   SEGMENT BEET ADDRESS (PTS TO OVB+1)
     +----------+----------+
 >12 !       SSBLEN        !   LENGTH OF SEGMENT (BYTES)
     +----------+----------+
 >14 !       SSBREC        !   REC. # OF PF SEG. ON HOME FILE
     +----------+----------+
 >16 !       SSBLOD        !   LOAD ADDRESS OF SEGMENT (FROM P.F.)
     +----------+----------+
 >18 !       SSBFLG        !   SEGMENT FLAGS
     +----------+----------+
 >1A !  SSBOVL  !  SSBPRI  !   LAST OVERLAY NUMBER LOADED IN SEG
     +----------+----------+      INSTALLED PRIORITY (TASKS ONLY)
 >1C !       SSBSTE        !   POINTER TO SWAP TABLE ENTRY
     +----------+----------+
 >1E !       SSBSOB        !   POINTER TO SEGMENT OWNER BLOCK
     +----------+----------+
 >20 !  SSBPRC  !  SSBPR2  !   THE ID'S OF PROCEDURES ASSOCIATED
     +----------+----------+      WITH THE TASK (TASK SEG ONLY)
 >22 !  SSBNAM  !          !   TASK NAME (TASK SEG ONLY)
     +----------+----------+
     /          /          /
     /          /          /
     +----------+----------+


 FLAGS FOR FIELD: SSBATR    #08 - SEGMENT ATTRIBUTES

   SSFRED = (X...............) - READABLE (NONTASK)
   SSFSYS = (.X..............) - SYSTEM (BOTH)
   SSFRES = (..X.............) - MEMORY RESIDENT (BOTH)
          = (...X............) - RESERVED
   SSFREP = (....X...........) - REPLICATABLE (BOTH)
   SSFSHR = (.....X..........) - SHARE PROTECT (NONTASK)
   SSFPR2 = (......X.........) - PROC 2 ON SYS P.F.(TASK)
          = (.......X........) - RESERVED
   SSFOVF = (........X.......) - OVERFLOW (TASK)
   SSFWCS = (.........X......) - WRITEABLE CONTROL STORE (BOTH)
   SSFEXC = (..........X.....) - EXECUTE PROTECT (BOTH)
   SSFWRT = (...........X....) - WRITE PROTECT (NONTASK)
   SSFUPD = (............X...) - UPDATEABLE (BOTH)
   SSFREU = (.............X..) - REUSEABLE (BOTH)
   SSFCPY = (..............X.) - COPYABLE (BOTH)
   SSFSEC = (...............X) - SECURITY BYPASS (TASK)


 FLAGS FOR FIELD: SSBFLG    #18 - SEGMENT FLAGS

   SSFTSK = (X...............) - TASK SEGMENT
   SSFEMP = (.X..............) - EMPTY SEGMENT (DO NOT LOAD)
   SSFHOM = (..X.............) - LOAD FROM HOME FILE
   SSFINI = (...X............) - INITIAL LOAD SEGMENT (SSB NOT
*                              INITIALIZED)
   SSFNRP = (....X...........) - DO NOT REPLICATE SSB (SINCE A
*                              GET MEMORY WAS DONE)
   SSFREL = (.....X..........) - RELEASABLE
   SSFMOD = (......X.........) - MODIFIED
   SSFMEM = (.......X........) - IN MEMORY
   SSFLLM = (........X.......) - LOGICALLY IN MEMORY


 EQUATES:

    LABEL   EQUATE TO   VALUE   DESCRIPTION
    -----   ---------   -----   ------------------------------
    SSBWCT   SSBID1       >02    UNALLOCATED WORDS IN SPECIAL TABLE
    SSFPRI   SSFRED       >00    PRIVILEGED (TASK)
    SSFPR1   SSFSHR       >05    PROC 1 ON SYS P.F.(TASK)
```

SSFSPR	SSFWRT	>0B	SOFTWARE PRIVILEGED (TASK)
SSBSIZ	$	>20	BASIC SSB SIZE
SSBTSZ	$	>2A	TASK SEGMENT SSB SIZE

```
***********************************************************
*                                                         *
*  SYSTEM TABLE AREA OVERHEAD  (STA)            01/20/79 *
*                                                         *
*            LOCATION: START OF ALL TABLE AREAS           *
***********************************************************
* THE STA DESCRIBES OVERHEAD INFORMATION AT THE START OF
* EACH OF THE SYSTEM TABLE AREAS: THE FILE MANAGEMENT TABLE
* AREA, THE BUFFER TABLE AREA, THE SEGMENT MANAGEMENT TABLE
* AREAS, AND THE STANDARD SYSTEM TABLE AREA.


      *----------+----------*
  >00 !       STAHED        !   FIRST ENTRY ON FREE MEMORY LST
      +----------+----------+
  >02 !       STALNK        !   POINTER TO FREE MEMORY CHAIN
      +----------+----------+
  >04 !       STARES        !   RESERVED TABLE AREA BOUNDRY
      +----------+----------+
  >06 !       STAEND        !   ENDING ADDRES OF TABLE AREA
      +----------+----------+
  >08 !       STAUSE        !   CURRENT TABLE USAGE
      +----------+----------+
  >0A !       STAHI         !   HIGHEST MEMORY ALLOCATION
      +----------+----------+
  >0C !       STAPTR        !   POINTER TO TABLE OWNER(JSB IF
      +----------+----------+      JCA OR SSB IF SPECIAL TABLE)

  EQUATES:

     LABEL   EQUATE TO   VALUE   DESCRIPTION
     -----   ---------   -----   ------------------------------
     STASIZ  $             >0E
```

```
************************************************************
*                                                          *
*  SWAP TABLE ENTRY                  (STE)   11/05/81      *
*                                                          *
*            LOCATION: SYSTEM JCA                          *
************************************************************
* FOR EACH SEGMENT ON THE SWAP FILE, THERE EXISTS AN STE
* IN MEMORY, LINKED IN FILE RECORD ORDER ON THE FILE.  THE
* ANCHOR OF STES IS ROLDIR IN PMDATA.


     *----------+----------*
 >00 !        STERDT       !   LINK TO NEXT STE
     +----------+----------+
 >02 !        STEPRC       !   ROLL FILE PHYS. RECORD NUMBER
     +----------+----------+
 >04 !        STERRL       !   NUMBER RECORDS IN ROLL FILE
     +----------+----------+
 >06 !        STELDT       !   CONTENTS OF OVBPTR
     +----------+----------+

 EQUATES:

    LABEL   EQUATE TO   VALUE   DESCRIPTION
    -----   ---------   -----   ------------------------------
    STESIZ  $             >08
```

```
**************************************************************
*                                                            *
*      TIME DELAY LIST ENTRY            (TDL)   03/15/78     *
*                                                            *
*           LOCATION:  SYSTEM TABLE AREA                     *
**************************************************************
* A TDL DESCRIBES AN ENTRY ON THE TIME DELAY LIST.


      *----------+----------*
  >00 !        TDLSVC       !   OP CODE (200)
      +----------+----------+
  >02 !        TDLTM1       !   REACTIVATION TIME (WD 1)
      +----------+----------+
  >04 !        TDLTM2       !
      +----------+----------+


  EQUATES:

     LABEL   EQUATE TO   VALUE   DESCRIPTION
     -----   ---------   -----   ------------------------------
     TDLVAL  $              >02    TIME DELAY VALUE
     TDLSIZ  $              >06    SIZE OF BLOCK
```

```
***********************************************************
*                                                         *
*  TASK STATUS BLOCK          (TSB)            04/04/83   *
*                                                         *
*            LOCATION:  JCA                               *
***********************************************************
* EACH TASK WHICH HAS BEEN BID IS REPRESENTED BY A TSB IN
* ITS JOB'S JCA.  THE TSB INCLUDES STATE INFORMATION, LINKS
* TO VARIOUS QUEUES, CHARACTERISTICS OF THE TASK, LOCATION
* INFORMATION, MAPPING INFORMATION, AND STATISTICS COUNTERS.
*
* DETAILS ABOUT PARTICULAR FIELDS:
*   TSBTSK - OFFSET INTO MAP FILE AND SSB ADDRESSES FOR THE
*            SEGMENT THAT IS THE TASK SEGMENT (0=FIRST SEGMENT,
*            4=SECOND SEGMENT, 8=THIRD SEGMENT)
*
*   TSBIOI - I/O BOUND INDICATOR, MODIFIED BY THE SCHEDULER
*
*   TSBGEN - GENERATION NUMBER IS ONE GREATER THAN THAT OF THE
*            PARENT OF THIS TASK.  IF THE PARENT TASK DIES, THE
*            GENERATION NUMBER OF THIS TASK IS REDUCED BY 1,
*            AS ARE THE GENERATION NUMBERS OF ANY DESCENDENTS
*            OF THIS TASK.  THIS VALUE IS 0 FOR QUEUE SERVERS.
*
*   TSBSBN - SMT AND SSB PAIR FOR THE NEW SEGMENT WHEN A CHANGE
*   TSBSTN   SEGMENT OPERATION IS ISSUED.  TSBPSN IS THE
*   TSPPSN   POSITION OF THE SEGMENT (0,4, OR 8).  THESE ARE
*            USED ONLY WHEN THE SSB FOR THE NEW SEGMENT MUST
*            BE INITIALIZED.
*
*   TSBLSE - LOAD SEGMENT ENTRIES INCLUDE THE JCA AND ANY OTHER
*            SEGMENTS THAT NEED TO BE LOADED IN MEMORY WHEN
*            THIS TASK EXECUTES, THOUGH THEY MAY NOT BE MAPPED
*            IN TO THE TASK.
*
*   TSBOSE - OWNED SEGMENTS ARE TEMPORARILY SHARE-PROTECTED
*
                          ** BEGINNING PACKED RECORD TSB


      *----------+----------*
  >00 !        TSBQL        !   QUEUEING LINK FOR DYNAMIC QUEUES
      +----------+----------+
  >02 !        TSBWP        !   ACTIVE WORKSPACE POINTER
      +----------+----------+
  >04 !        TSBPC        !   ACTIVE PROGRAM COUNTER
      +----------+----------+
  >06 !        TSBST        !   ACTIVE STATUS
      +----------+----------+
  >08 !  TSBPRI  !  TSBSTA  !   TASK PRIORITY (RUN TIME)
      +----------+----------+      TASK STATE
  >0A !  TSBIPR  !  TSBINP  !   INITIAL TASK PRIORITY
```

```
    +----------+----------+      INSTALLED TASK PRIORITY
>0C ! TSBIID   !  TSBRID  !   INSTALLED TASK IDENTIFIER
    +----------+----------+      RUN TIME TASK IDENTIFIER
>0E ! TSBSTG   !  TSBIEC  !   TASK STAGE NUMBER
    +----------+----------+      INITIATED EVENT COUNT
>10 !        TSBFL1       !   TASK FLAGS - SYSTEM FLAGS
    +----------+----------+
>12 !        TSBFL2       !   TASK FLAGS - CONTROL FLAGS
    +----------+----------+
>14 !        TSBJSB       !   JSB ADDRESS
    +----------+----------+
>16 !        TSBXOP       !   EFFECTIVE XOP ADDRESS
    +----------+----------+
>18 ! TSBTSK   !  TSBIO   !   2 WORD OFFSET TO TASK (0,4,8)
    +----------+----------+      GENERAL I/O COUNT
>1A ! TSBIOI   !  TSBPSN  !   I/O BOUND INDICATOR
    +----------+----------+      POSITION OF NEW SSB
>1C !        TSBCPT       !   CPU EXECUTION TIME (TICKS)
    +----------+----------+
>1E !                     !
    +----------+----------+
>20 !        TSBRPC       !   NUMBER SERVICE CALLS ISSUED
    +----------+----------+
>22 !                     !
    +----------+----------+
>24 !        TSBBY1       !   NUMBER I/O BYTES TRANSFERRED
    +----------+----------+
>26 !        TSBBY2       !   I/O BYTES TRANSFERRED - WORD 2
    +----------+----------+
>28 !        TSBSPN       !   TICK COUNTER AT TIME SUSPENDED
    +----------+----------+
>2A !        TSBTSB       !   TSB FIXED LINK IN SET FOR JOB
    +----------+----------+
>2C ! TSBSTI   !  TSBGEN  !   STATION ID (FF=NO STATION)
    +----------+----------+      GENERATION NUMBER
>2E !        TSBPM1       !   PARAMETER 1
    +----------+----------+
>30 !        TSBPM2       !   PARAMETER 2
    +----------+----------+
>32 !        TSBIN1       !   COMPLETED EVENT FLAGS - WORD 1
    +----------+----------+
>34 !        TSBIN2       !   COMPLETED EVENT FLAGS - WORD 1
    +----------+----------+
>36 !        TSBLDT       !   LDT LIST HEADER POINTER
    +----------+----------+
>38 !        TSBEOR       !   END OF REQUEST PROCESSING
    +----------+----------+      LIST HEADER
>3A !        TSBEAP       !   END ACTION PROGRAM COUNTER
    +----------+----------+
>3C !        TSBEAW       !   END ACTION WORKSPACE
    +----------+----------+
>3E !        TSBDIA       !   END ACTION STATUS INFORMATION
    +----------+----------+      (DIAGNOSTIC DATA ADDRESS)
```

```
>40 !        TSBSBN         !   ADDRESS OF NEW SSB
    +-----------+-----------+
>42 !        TSBSTN         !   SM TABLE SSB FOR NEW SEGMENT
    +-----------+-----------+
>44 !        TSBSB1         !   SSB ADDRESS FOR 1ST SEGMEMT
    +-----------+-----------+
>46 !        TSBST1         !   SM TABLE SSB FOR 1ST SEGMENT
    +-----------+-----------+
>48 !        TSBSB2         !   SSB ADDRESS FOR 2ND SEGMEMT
    +-----------+-----------+
>4A !        TSBST2         !   SM TABLE SSB FOR 2ND SEGMENT
    +-----------+-----------+
>4C !        TSBSB3         !   SSB ADDRESS FOR 3RD SEGMENT
    +-----------+-----------+
>4E !        TSBST3         !   SM TABLE SSB FOR 3RD SEGMENT
    +-----------+-----------+
>50 !        TSBML1         !   MAP LIMIT ONE REGISTER
    +-----------+-----------+
>52 !        TSBMB1         !   MAP BIAS ONE REGISTER
    +-----------+-----------+
>54 !        TSBML2         !   MAP LIMIT TWO REGISTER
    +-----------+-----------+
>56 !        TSBMB2         !   MAP BIAS TWO REGISTER
    +-----------+-----------+
>58 !        TSBML3         !   MAP LIMIT THREE REGISTER
    +-----------+-----------+
>5A !        TSBMB3         !   MAP BIAS THREE REGISTER
    +-----------+-----------+
>5C !        TSBBLN         !   LENGTH OF MAPPED SEGMENTS (BEETS)
    +-----------+-----------+
>5E !        TSBTLM         !   TOTAL ROLLABLE MEMORY (BEETS)
    +-----------+-----------+
>60 !        TSBMXM         !   MAX VALUE OF TSBTLM
    +-----------+-----------+
>62 !        TSBLSE         !   LOAD SEGMENT ENTRY LIST HEADER
    +-----------+-----------+
>64 !        TSBOSE         !   OWNED SEGMENT ENTRY LIST HEADER
    +-----------+-----------+
>66 !        TSBSRT         !   TICK COUNTER WHEN TASK STARTED
    +-----------+-----------+
>68 !                       !
    +-----------+-----------+
>6A !        TSBFMT         !   THE FDP OF PROGRAM FILE FOR
    +-----------+-----------+
>6C !        TSBFCB         !   THE TASK SEGMENT
    +-----------+-----------+
>6E !   TSBAIC  !   TSBRES  !   ABORTING I/O COUNT
    +-----------+-----------+      RESERVED
>70 SIZE                 ** END OF PACKED RECORD


 FLAGS FOR FIELD: TSBFL1     #10 - TASK FLAGS - SYSTEM FLAGS
```

```
    TSFSYS = (X...............) - SYSTEM TASK
    TSFPRI = (.X..............) - PRIVILEGED TASK
    TSFMEM = (..X.............) - CURRENT SEGMENT SET IN MEMORY
    TSFENA = (...X............) - TAKE END ACTION ON ERROR
    TSFIOA = (....X...........) - I/O HAS BEEN ABORTED FOR TASK
    TSFABT = (.....X..........) - TASK BEING ABORTED
    TSFSEC = (......X.........) - BYPASS SECURITY
    TSFQSR = (.......X........) - QUEUE SERVER TASK
    TSFACT = (........X.......) - ACTIVATE TASK OUTSTANDING
    TSFBID = (.........X......) - INITIAL TASK BID
    TSFSPR = (..........X.....) - SOFTWARE PRIVILEGE
    TSFTOA = (...........X....) - ABORT TIMEOUT FLAG
    TSFIOE = (............X...) - I/O EVENT PEND. UNBUFF
*                                  RESERVED - BITS 13 - 15

  FLAGS FOR FIELD: TSBFL2    #12 - TASK FLAGS - CONTROL FLAGS

    TSFCNT = (X...............) - TASK BEING CONTROLLED
    TSFSSC = (.X..............) - STOPPED BY SCHEDULER
    TSFSBK = (..X.............) - STOPPED BY BREAKPOINT
    TSFHLT = (...X............) - TASK TO BE HALTED
    TSFRST = (....X...........) - RESTART PARENT ON TERM
    TSFRBD = (.....X..........) - RBID TASK
    TSFXOP = (......X.........) - REISSUE XOP
    TSFCHO = (.......X........) - JOB-LOCAL CHANNEL OWNER
*                                  RESERVED - BITS 8 - 15
```

```
***********************************************************
*                                                         *
*  USER DESCRIPTOR OVERFLOW RECORD  (UDO)   11/24/82      *
*                                                         *
*           LOCATION: S$CLF ON DISK                       *
***********************************************************
* THE UDO IS USED ONLY IN THE CASE THAT A USER IS A MEMBER
* OF MORE ACCESS GROUPS THAN WILL FIT IN HIS UDR.  IT CONTAINS
* ACCESS GROUP INFORMATION.  IT IS A VARIANT OF THE CAPABILITIES
* LIST FILE RECORD (CLR).  FOR DETAILS SEE CLR.
```

```
***********************************************************
*                                                         *
*  USER DESCRIPTOR RECORD      (UDR)        11/24/82      *
*                                                         *
*          LOCATION: DISK                                 *
***********************************************************
* THE UDR DESCRIBES THE DISK STRUCTURES THAT REPRESENTS A
* GIVEN USER OF THE SYSTEM.  IT INCLUDES LOGON INFORMATION
* AND SECURITY INFORMATION.  IT IS A VARIANT OF THE CAPABILITIES
* LIST FILE RECORD (CLR).  FOR DETAILS SEE CLR.
```

```
*************************************************************
*                                                           *
*  USER ID PARAMETER (UIP)                         12/01/82 *
*                                                           *
*      LOCATION: POINTED TO BY IRBPRM FIELD                 *
*                                                           *
*************************************************************
*
*      THE USER ID PARAMETER IS CHECKED BY SECURITY
*      MANAGER AND WILL BE USED IN PLACE OF THE
*      ISSUER'S USER ID IF A VALID PASSCODE IS
*      SUPPLIED OR THE TASK HAS SECURITY BYPASS.
*      THIS PARM MAY BE SUPPLIED BY A USER, OR
*      MAY BE CREATED BY IOU TO PASS INFO ACROSS
*      THE NETWORK.
*
```

```
      *----------+----------*
  >00 !  UIPSLN  !  UIPLEN  !   SUBLIST NUMBER (>02)
      +----------+----------+      LENGTH OF PARM-2 IN BYTES (>10)
  >02 !  UIPUID  !          !   USER ID
      +----------+----------+
      /          /          /
      /          /          /
      +----------+----------+
  >0A !  UIPPWD  !          !   PASSWORD
      +----------+----------+
      /          /          /
      /          /          /
      +----------+----------+
```

EQUATES:

LABEL	EQUATE TO	VALUE	DESCRIPTION
UIPSIZ	$	>12	TOTAL LENGTH OF UIP

```
*************************************************************
*      VALUE CONTINUATION BLOCK    (VCB)           07/16/81  *
*                                                           *
*      LOCATION:  A NAME DEFINITION SEGMENT                 *
*************************************************************


      *----------+----------*
  >00 !       VCBVCB        !   POINTER TO NEXT NCB
      +----------+----------+

  EQUATES:

     LABEL   EQUATE TO   VALUE   DESCRIPTION
     -----   ---------   -----   ------------------------------
     VCBSIZ  $             >02     LENGTH OF NCB OVERHEAD
```

```
***************************************************
*      VALUE DEFINITION BLOCK  (VDB)           07/21/81
*
*    LOCATION:  NAME DEFINITION SEGMENT
***************************************************


     *----------+----------*
 >00 !        VDBVCB       !   NEXT NAME CONTINUATION BLOCK
     +----------+----------+
 >02 !  VDBUSE  !          !   NUMBER OF USERS OF THIS VALUE
     +----------+----------+

 EQUATES:

    LABEL   EQUATE TO   VALUE   DESCRIPTION
    -----   ---------   -----   ------------------------------
    VDBSIZ   $            >03     SIZE OF THE VDB OVERHEAD
```

```
*****************************************************************
*                                                               *
*   VIRTUAL REQUEST BLOCK      (VRB)              8/22/83       *
*                                                               *
*             LOCATION: SYSTEM TABLE AREA AND JCA               *
*                                                               *
*****************************************************************
*  DEFINITIONS OF FIELDS IN DSR CALL BLOCK (DATA BUFFER FOR I/O
*  SVC SUBOPCODE >17) FOR VIRTUAL TERMINAL DSR.
*
                          ** BEGINNING PACKED RECORD VRB


      *----------+----------*
  >00 !  VRBOC   !  VRBEC   !   VTDSR REQUEST CODE
      +----------+----------+      VTDSR RETURN CODE
  >02 !        VRBVT        !   VIRTUAL TERMINAL NUMBER (HEX)
      +----------+----------+
  >04 !  VRBCHR  !  VRBLC   !   BID CHAR
      +----------+----------+      LUNO USE COUNT
  >06 !        VRBJOB       !   PDTJOB
      +----------+----------+
  >08 !        VRBRDN       !   REMOTE DEVICE NAME
      +----------+----------+
      /          /          /
      /          /          /
      +----------+----------+
  >16 !        VRBJID       !   REAL TERM JOB I.D.
      +----------+----------+
  >18 !        VRBIPC       !   OWNER IPC NAME
      +----------+----------+
      /          /          /
      /          /          /
      +----------+----------+
  >24 SIZE                ** END OF PACKED RECORD
```

```
*************************************************************
*                                                           *
*  EXTENTION FOR A TERMINAL   (XTK)              02/12/82   *
*          WITH A KEYBOARD                                  *
*                                                           *
*          LOCATION: SYSTEM AREA                            *
*************************************************************
* THE XTK IS AN EXTENSION TO THE PDT USED TO DESCRIBE A
* DEVICE WITH A KEYBOARD.  IT IS USED AS A WORK AREA BY
* THE DSR.


      *----------+----------*
  >00 !        XTKXUF       !   EXTENDED USER FLAGS FROM BRB
      +----------+----------+
  >02 !  XTKFLG  !  XTKSCH  !   XTK GENERAL FLAGS
      +----------+----------+      SAVED CHAR FOR JISCII TERMINAL
  >04 !        XTKCRD       !   CARRIAGE RETURN DELAY COUNT
      +----------+----------+
  >06 !        XTKICD       !   INTER-CHARACTER DELAY COUNT
      +----------+----------+
  >08 !        XTKSSC       !   SAVED STATUS OF CASSETTES
      +----------+----------+
  >0A !        XTKABT       !   CODE ADDRESS TO PERFORM ABORT
      +----------+----------+
  >0C !        XTKTMO       !   TIME-OUT COUNT FOR HANG CONDITION
      +----------+----------+
  >0E !        XTKPFR       !   POWER FAIL FLAG/BUFFER BIAS
      +----------+----------+
  >10 !        EDTFL0       !   EXTENDED EDIT FLAGS - WORD 0
      +----------+----------+
  >12 !        EDTFL1       !   EXTENDED EDIT FLAG - WORD 1
      +----------+----------+


  FLAGS FOR FIELD: XTKFLG    #02 - XTK GENERAL FLAGS

    KSFHNG = (X...............) - HANG UP CONDITION ON 745
    KSFTMS = (.X..............) - TIME-OUT SWITCH FOR 745
    KSFSCI = (..X.............) - SCI ACTIVE DURING HANG UP
    KSFDCD = (...X............) - DATA CARRIER DROP DETECTED
    KSFSIO = (....X...........) - SHIFT IN/SHIFT OUT JISCII
    KSFDIF = (.....X..........) - DIRECT CHAR INPUT REQUESTED


  FLAGS FOR FIELD: EDTFL0    #10 - EXTENDED EDIT FLAGS - WORD 0

             = (X...............) - PASS-THROUGH MODE
             = (.X..............) - 940-IN PTM, TERMINATE READ ON ETX
             = (..X.............) - 940-IN PTM, TERMINATE READ ON ESC-)
             = (...X............) - USED, BUT NOT DOCUMENTED
             = (....X...........) - 940-DISABLE USE OF BIT 0 FOR INTEN
```

```
         = (.....X..........) - 940-ALLOW ESC & SOH IN WRITE ASCII B
         = (......X.........) - 940-IGNORE DISPLAY CHARACTERS, ETC.
         = (.......X........) - 940-1=132 COL MODE; 0=80 COL MOD
  MDTCHK = (........X.......) - POST DATA MODIFIED ON READ
  EXVAL  = (.........X......) - EXTENDED CHAR VALIDATION
  NULFLG = (..........X.....) - NULL CHARACTER SUPPRESSION
  CNBFLG = (...........X....) - CONVERT NULL TO BLANK
         = (............X...) - KANJI
         = (.............XXX) - RESERVED


FLAGS FOR FIELD: EDTFL1    #12 - EXTENDED EDIT FLAG - WORD 1

         = (X...............) - TERMINATE READ ON ERASE FIELD
         = (.X..............) - TERMINATE READ ON RIGHT FIELD
  LEFARO = (..X.............) - TERMINATE READ ON LEFT ARROW
         = (...X............) - TERMINATE READ ON TAB
         = (....X...........) - TERMINATE READ ON UP ARROW
         = (.....X..........) - TERMINATE READ ON SKIP
         = (......X.........) - TERMINATE READ ON HOME
         = (.......X........) - TERMINATE READ ON RETURN
         = (........X.......) - TERMINATE READ ON ERASE INPUT
         = (.........X......) - TERMINATE READ ON BLANK GRAY
         = (..........X.....) - TERMINATE READ ON DELETE CHAR
         = (...........X....) - TERMINATE READ ON INSERT CHAR
  RITARO = (............X...) - TERMINATE READ ON RIGHT ARROW
         = (.............X..) - TERMINATE READ ON ENTER
         = (..............X.) - TERMINATE READ ON LEFT FIELD
         = (...............X) - TERMINATE READ ON DOWN ARROW
*
```

EQUATES:

LABEL	EQUATE TO	VALUE	DESCRIPTION
XTKFIL	XTKFLG	>02	FILL CHARACTER
XTKEVT	XTKFLG+1	>03	EVENT CHARACTER
XTKPOS	XTKCRD	>04	WITHIN FIELD CURSOR POSITION
XTKDEF	XTKICD	>06	START OF FIELD CURSOR POSITION
XTKJIN	XTKSSC	>08	ASCII/JISCII INTENSITY MASK
XTKSC1	XTKABT	>0A	SCRATCH # 1
XTKSC2	XTKTMO	>0C	SCRATCH # 2
XTKSIZ	$	>14	
XTKBUF	XTKSIZ	>14	CHARACTER BUFFER

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.

2274834 (1/14)

Table A-1. Generic Keycap Names

Generic Name	911 VDT	940 EVT	931 VDT	Business System Terminal	820[1] KSR
Alternate Mode	None	ALT	ALT	ALT	None
Attention[2]	[redacted]	SHIFT PREV FORM NEXT	▬	[redacted]	CTRL S
Back Tab	None	SHIFT ← TAB →	SHIFT ⇧ ⇤ TAB ⇥	None	CTRL T
Command[2]	[redacted]	PREV FORM NEXT	CMD	[redacted]	CTRL X
Control	CONTROL	CTRL	CTRL	CTRL	CTRL
Delete Character	DEL CHAR	LINE DEL CHAR	DEL CHAR	DEL CHAR	None
Enter	[redacted]	SEND	ENTER	ENTER	CTRL Y
Erase Field	ERASE FIELD	EOS ERASE EOF	ERASE FIELD	ERASE FIELD	CTRL] [

Notes:

[1] 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.

[2] On a 915 VDT the Command Key has the label F9 and the Attention Key has the label F10.

2284734 (2/14)

Table A-1. Generic Keycap Names (Continued)

Generic Name	911 VDT	940 EVT	931 VDT	Business System Terminal	820[1] KSR
Erase Input	ERASE INPUT	ALL ERASE INPUT	ERASE INPUT	ERASE INPUT	CTRL N
Exit	ESC	PREV PAGE NEXT	SHIFT ⇧ ESC	SHIFT ESC	ESC
Forward Tab	SHIFT TAB SKIP	← TAB →	⇤ TAB ⇥	SHIFT TAB SKIP	CTRL I
F1	F1	F1	F1	F1	CTRL A
F2	F2	F2	F2	F2	CTRL B
F3	F3	F3	F3	F3	CTRL C
F4	F4	F4	F4	F4	CTRL D

Notes:

[1] 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.

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	F5	F5	F5	F5	CTRL E
F6	F6	F6	F6	F6	CTRL F
F7	F7	F7	F7	F7	CTRL V
F8	F8	F8	F8	F8	CTRL W
F9	CONTROL ! 1	F9	F9	SHIFT F1	CTRL] [
F10	CONTROL @ 2	F10	F10	SHIFT F2	CTRL Z

Notes:

[1] 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.

2284734 (4/14)

Table A-1. Generic Keycap Names (Continued)

Generic Name	911 VDT	940 EVT	931 VDT	Business System Terminal	820[1] KSR
F11	CONTROL $ 4	F11	F11	SHIFT F3	CTRL ¦ \
F12	CONTROL % 5	F12	F12	SHIFT F4	CTRL } {
F13	CONTROL ^ 6	SHIFT F1	SHIFT ⇧ F1	SHIFT F5	CTRL + =
F14	CONTROL & 7	SHIFT F2	SHIFT ⇧ F2	SHIFT F6	CTRL — –
Home	HOME	HOME	HOME	HOME	CTRL L
Initialize Input	□	SHIFT LINE INS CHAR	□	■	CTRL O

Notes:

[1] 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.

2284734 (5/14)

Table A-1. Generic Keycap Names (Continued)

Generic Name	911 VDT	940 EVT	931 VDT	Business System Terminal	820[1] KSR
Insert Character	INS CHAR	LINE INS CHAR	INS CHAR	INS CHAR	None
Next Character	→ or SHIFT → CHAR ←	→	►	→	None
Next Field	SHIFT → FIELD ←	LINE FEED	SHIFT ⇧ → FIELD ←	SHIFT ← FIELD →	None
Next Line	↓	↓	▼	↓	CTRL J or LINE FEED
Previous Character	← or → CHAR ←	←	◄	←	None
Previous Field	→ FIELD ←	SHIFT ← SKIP →	→ FIELD ←	← FIELD →	None

Notes:
[1] 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.

2284734 (6/14)

Table A-1. Generic Keycap Names (Continued)

Generic Name	911 VDT	940 EVT	931 VDT	Business System Terminal	820[1] KSR
Previous Line	↑	↑	▲	↑	CTRL U
Print	PRINT	PRINT	PRINT	PRINT	None
Repeat	REPEAT	See Note 3	See Note 3	See Note 3	None
Return	[redacted]	[redacted]	↵ RETURN	[redacted]	[redacted]
Shift	SHIFT	SHIFT	SHIFT ⇧	SHIFT	SHIFT
Skip	TAB SKIP	← SKIP →	SKIP	TAB SKIP	None
Uppercase Lock	UPPER CASE LOCK	UPPER CASE	CAPS LOCK	UPPER CASE LOCK	UPPER CASE

Notes:

[1] 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.

[3] The keyboard is typamatic, and no repeat key is needed.

2284734 (7/14)

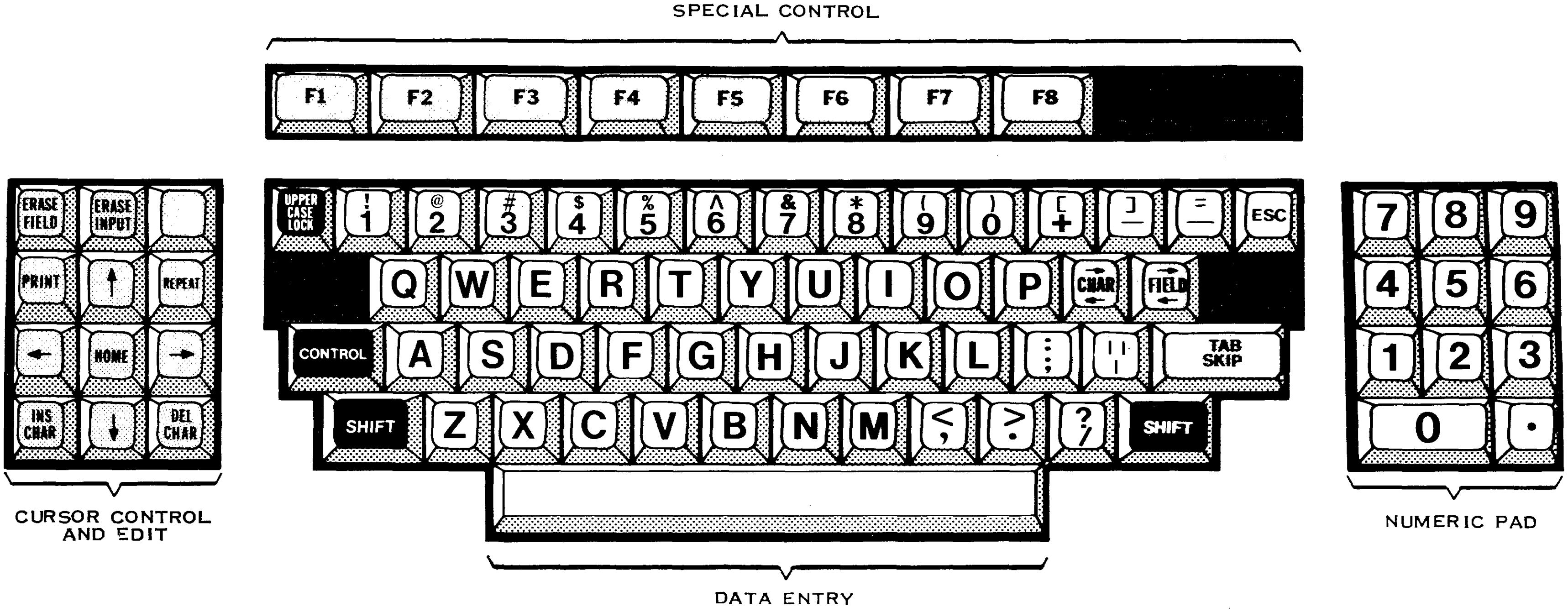
Table A-2. Frequently Used Key Sequences

Function	Key Sequence
Log-on	Attention/(Shift)!
Hard-break	Attention/(Control)x
Hold	Attention
Resume	Any key

Table A-3. 911 Keycap Name Equivalents

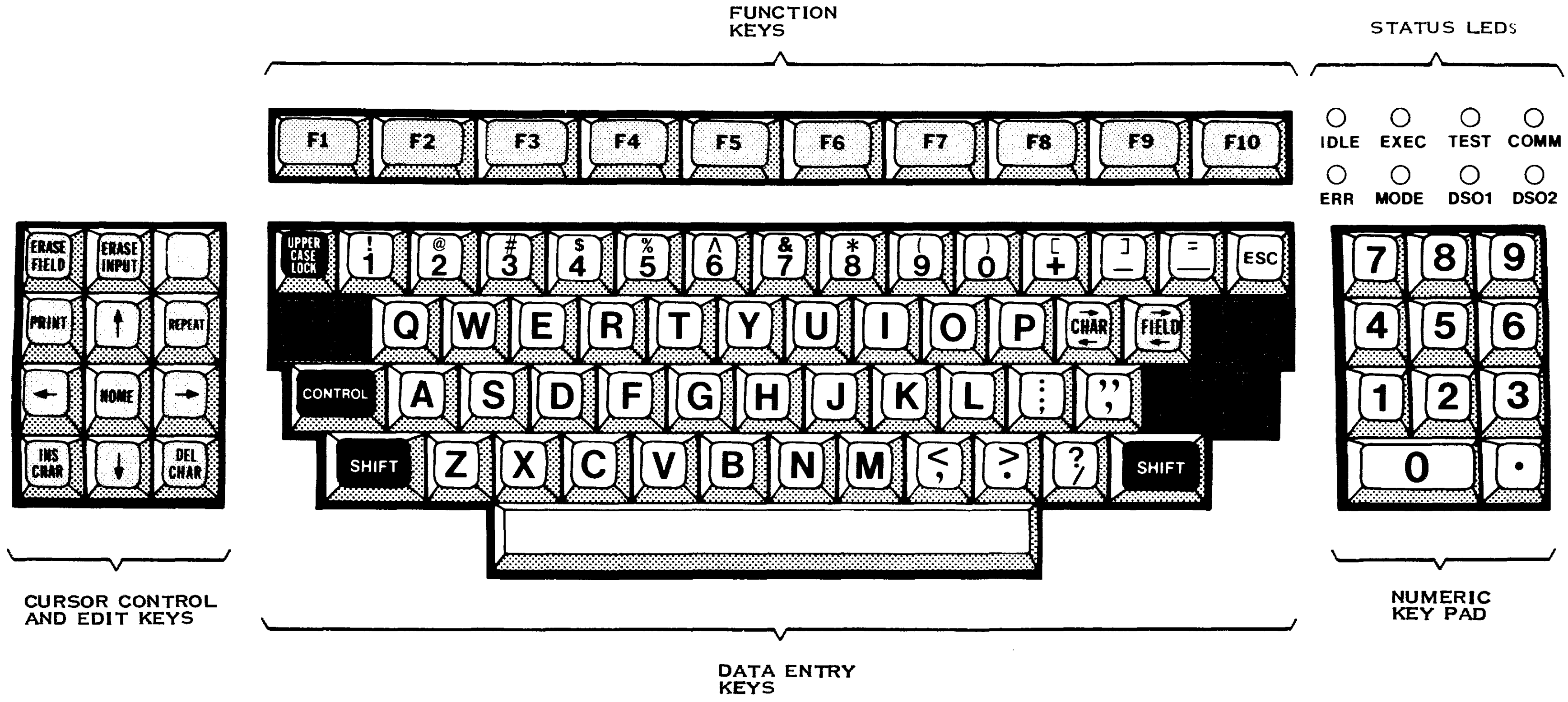
911 Phrase	Generic Name
Blank gray	Initialize Input
Blank orange	Attention
Down arrow	Next Line
Escape	Exit
Left arrow	Previous Character
Right arrow	Next Character
Up arrow	Previous Line

2284734 (8/14)



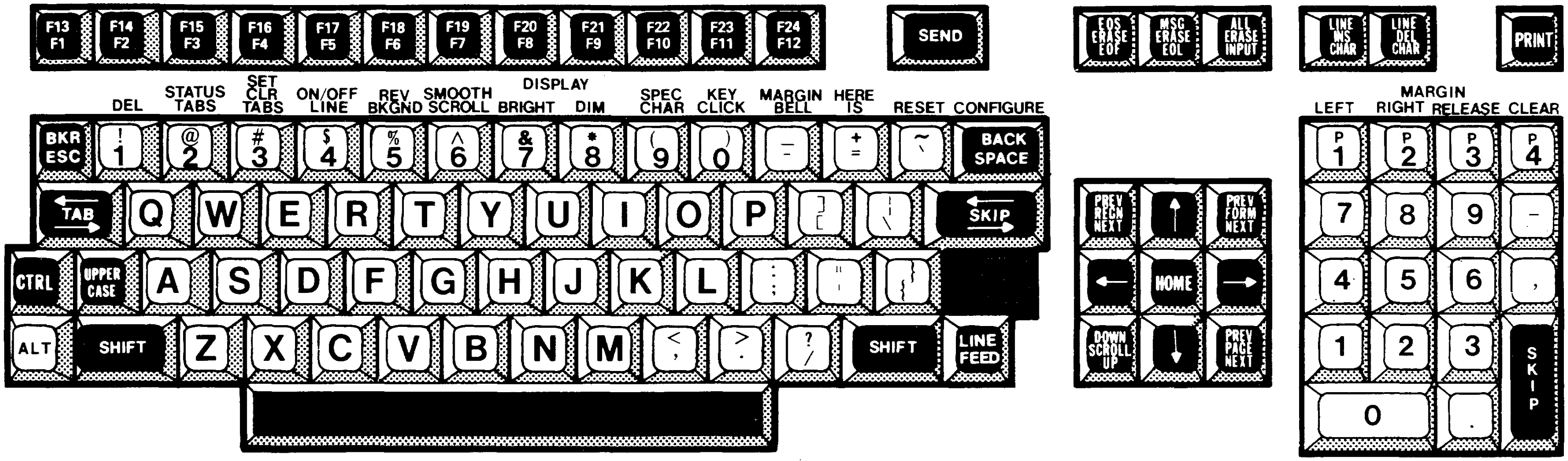
2284734 (9/14)

Figure A-1. 911 VDT Standard Keyboard Layout



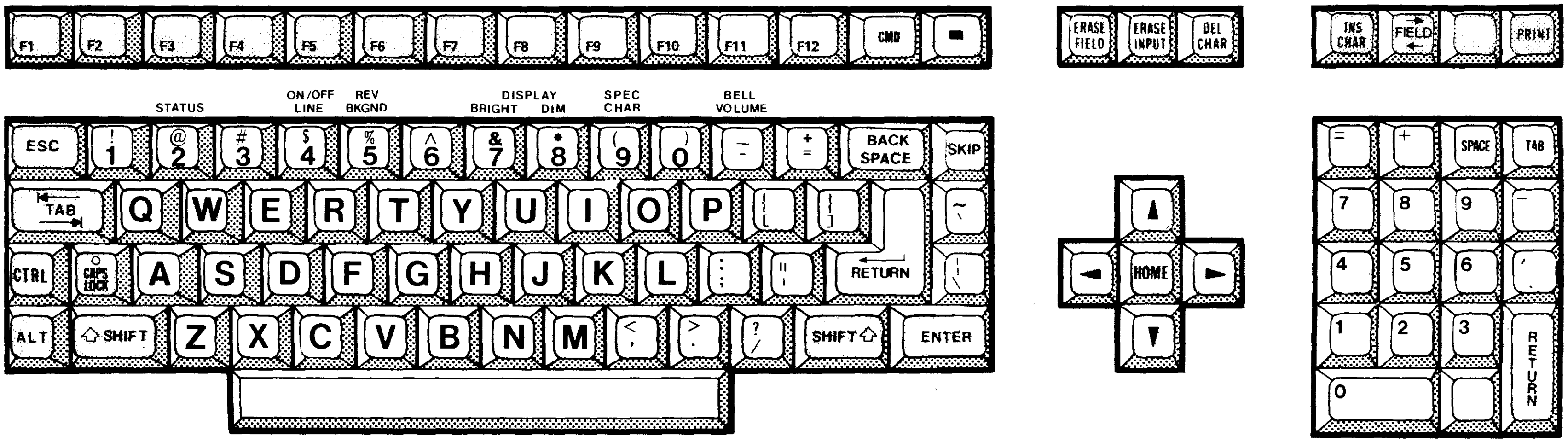
2284734 (10/14)

Figure A-2. 915 VDT Standard Keyboard Layout



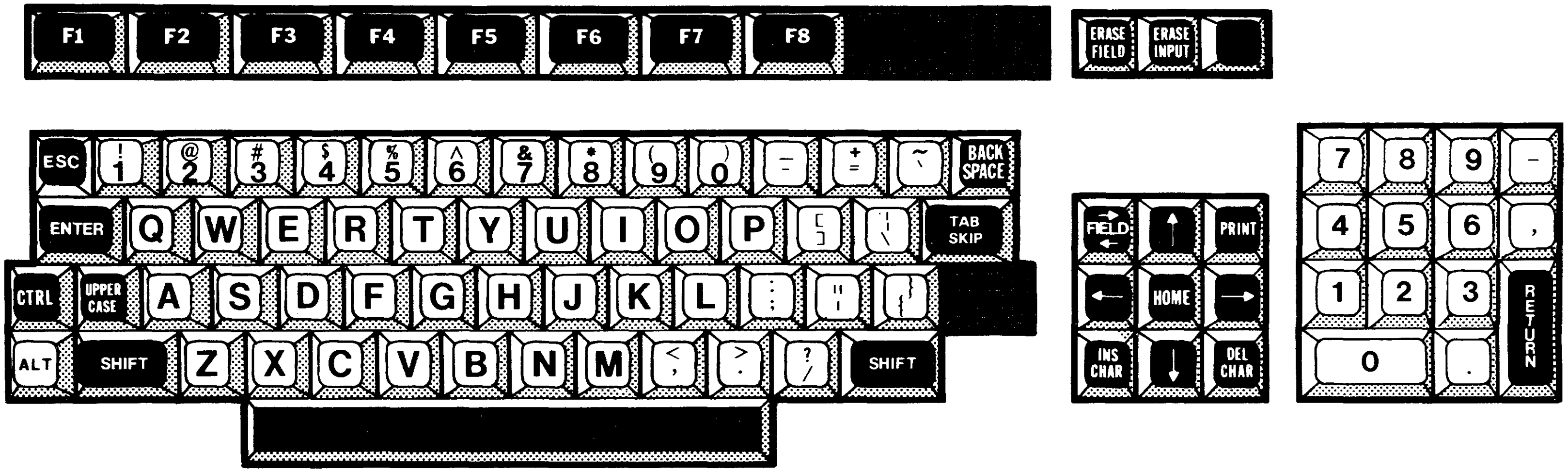
2284734 (11/14)

Figure A-3. 940 EVT Standard Keyboard Layout



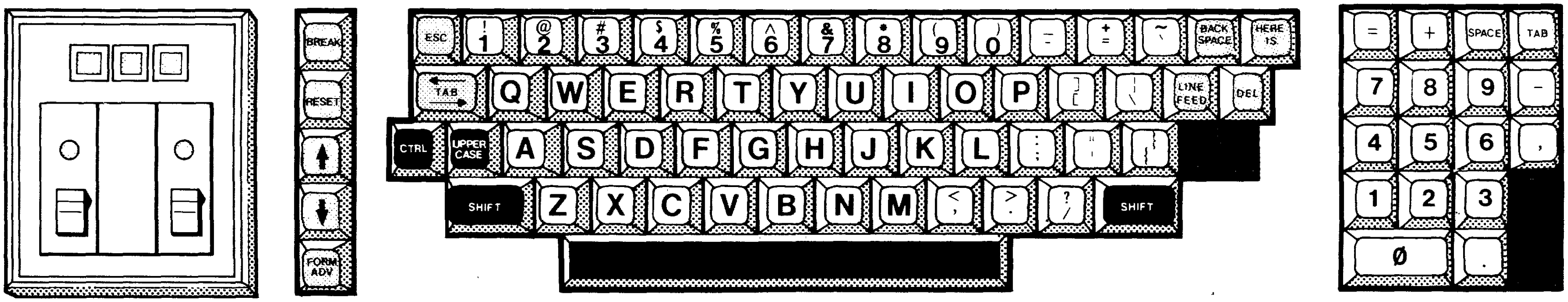
2284734 (12/14)

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

The following index lists key words and concepts from the subject material of this manual together with the area(s) in the manual that supply coverage of the listed concept. The numbers along with the right side of the listing reference the following manual areas:

- * Sections -- References to Sections of the manual appear as "Section x" with the symbol x reresenting any numeric quantity.

- * Appendixes -- References to Appendixes of the manual appear as "Appendix y" with the symbol y representing any capital letter.

- * Paragraphs -- References 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 chartacter refers to the section or appendix of the manual in which the paragraph is 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

Should you be unable to find the item of interest in the index, review the Table of Contents, List of Tables and List of Figures for general categories of information.

Alphabetical Index

CUT ALONG LINE

USER'S RESPONSE SHEET

Manual Title: DNOS System Design Document (2270512-9701)

Manual Date: 15 November 1983 **Date of This Letter:**

User's Name: **Telephone:**

Company: **Office/Department:**

Street Address:

City/State/Zip Code:

Please list any discrepancy found in this manual by page, paragraph, figure, or table number in the following space. If there are any other suggestions that you wish to make, feel free to include them. Thank you.

Location in Manual	Comment/Suggestion

NO POSTAGE NECESSARY IF MAILED IN U.S.A.
FOLD ON TWO LINES (LOCATED ON REVERSE SIDE), TAPE AND MAIL

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