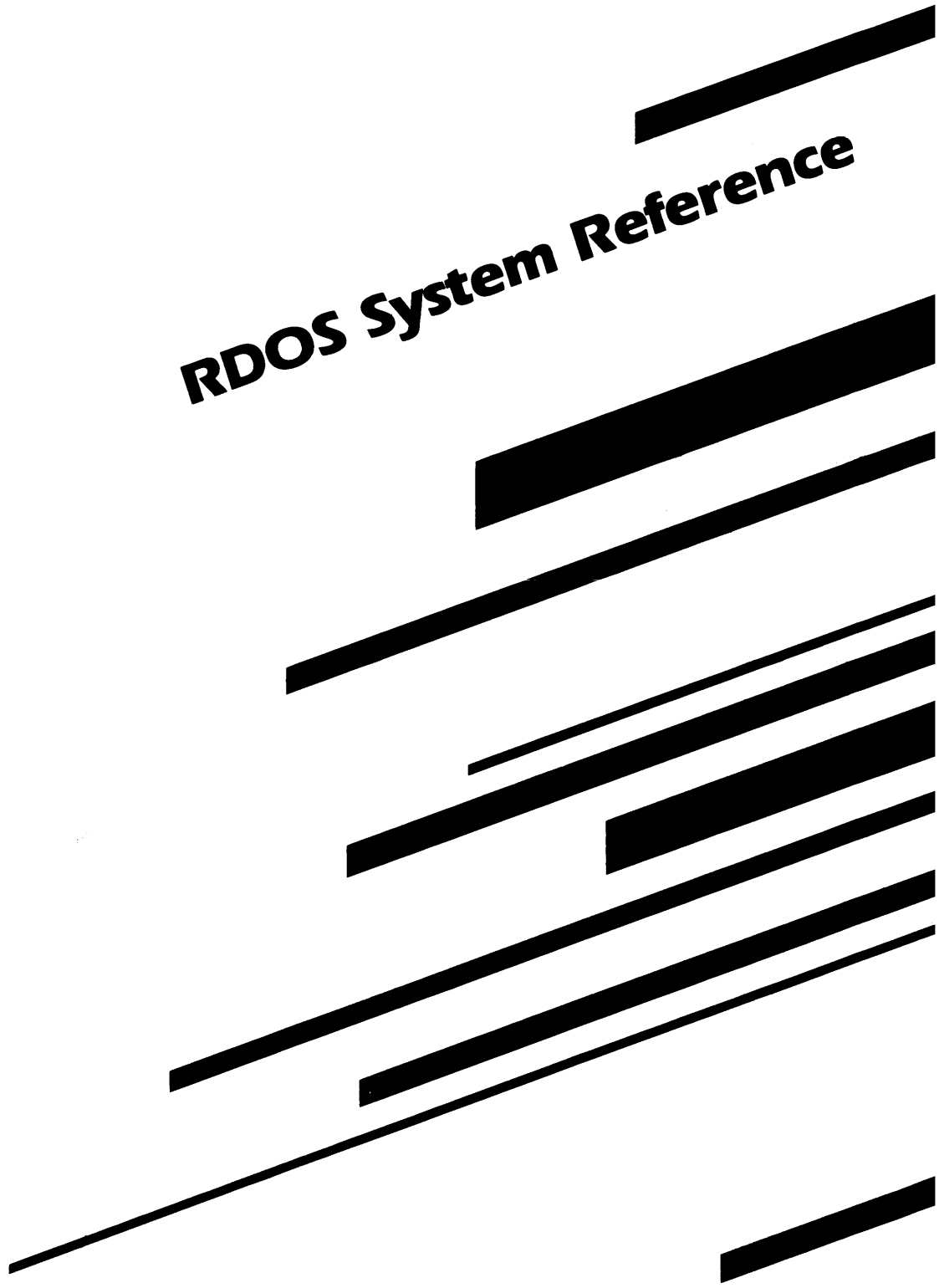
DataGeneral

RDOS System Reference



DataGeneral

RDOS System Reference

Notice

Revision History:

093-000075
Original Release - June 1972
First Revision - November 1972
Second Revision - May 1973
Third Revision - August 1973
Fourth Revision - December 1973
Fifth Revision - April 1974
Sixth Revision - January 1975
Seventh Revision - September 1975
Eighth Revision - March 1979

017-000002
Original Release - September 1972
First Revision - November 1972
Second Revision - January 1974
Third Revision - February 1975

093-000231 (Change in part number only from 017-000002-03)
Original Release - March 1979

093-400027
Original Release - October 1983

This manual (093-400027-00) supersedes the *Real Time Disk Operating System (RDOS) Reference Manual* (093-000075-08) and the *User Device Driver Implementation in the Real-Time Disk Operating System (RDOS)* manual (093-000231-00).

Ordering No. 093-400027

Rev. 00, October 1983

Preface

This is the primary reference manual for Data General's Real-Time Disk Operating System, RDOS. It describes all features of the operating system for the NOVA and ECLIPSE computers that it supports, and covers the salient points of dual programming for mapped machines. Much of the manual concerns system and task calls that can be used in assembly language programs. Users who intend to program in a higher-level language, such as FORTRAN, will find this book most helpful in conjunction with the manual that specifically describes that language.

Reading Path

This manual assumes (1) a basic understanding of RDOS features and concepts, (2) a currently running system that has been tailored to the user's hardware and software needs, and (3) a working knowledge of the RDOS/DOS Command Line Interpreter (CLI) program. First-time users of RDOS are strongly urged to consult the following manuals before this one.

Introduction to RDOS (DGC No. 069-400010)familiarizes readers with RDOS concepts and capabilities. The manual describes the Command Line Interpreter (CLI), the RDOS file system, input/output, memory organization and management, and special uses of RDOS such as user device drivers and multiple-processor systems. The manual also introduces RDOS utilities, providing practical examples in some cases to demonstrate their use.

How to Load and Generate RDOS (DGC No. 069-400013) steps readers through the procedures of program loading, disk initialization, installing the bootstrap root and RDOS starter system, and generating an RDOS system that meets their particular needs.

RDOS/DOS Command Line Interpreter (DGC No. 069-400015) describes the features and commands of the CLI—the primary interface between RDOS and its users. Among their many functions, CLI commands enable the user to create and protect files, and to invoke such utility programs as assemblers and text editors.

RDOS/DOS User's Handbook (DGC No. 069-400018) provides a handy summary of all CLI, utility, and RDOS commands.

Organization

This manual contains ten chapters and nine appendices, as follows.

Chapter 1, "Overview," introduces RDOS and explains how it runs in memory.

Chapter 2, "Files and Directories," explains RDOS files and file access, including file types, access modes, directories, linking, magnetic tape files, and multiplexors.

Chapter 3, "Single-task Programming," describes most of the system calls needed to program RDOS in a single-task environment. It summarizes the most commonly used system calls in table form.

Chapter 4, "Extending User Address Space," explains certain tools for extending addressable memory space, among them, program swaps, chains, user and virtual overlays, and window mapping.

Chapter 5, "Multitask Programming," explains the procedure of creating tasks within one program to manage diverse, real-time requirements.

Chapter 6, "Foreground-Background Programming," outlines the technique of running two programs simultaneously—in the foreground and background—on mapped and unmapped machines.

Chapter 7, "Interrupts and Power Failures," assists users who want to write their own interrupt handlers or define nonstandard devices in their RDOS systems.

Chapter 8, "Multiple Processor Systems," covers multiprocessor programming for those with more than one CPU who want to implement communications between them.

Chapter 9, "System Tuning," describes how RDOS uses stacks, cells, and buffers, and how the RDOS tuning feature checks and improves a system's performance.

Chapter 10, "Running in LEF Mode," explains for users with mapped ECLIPSE computers how to use the Load Effective Address (LEF) instruction.

Appendix A summarizes all RDOS system calls, task calls, and error messages.

Appendix B lists the source file PARU.SR, which describes all RDOS user parameters. This listing helps the user build program tables and understand how RDOS operates.

Appendix C demonstrates real-time programming with two examples.

Appendix D describes the directory RDOS uses to manage overlays.

Appendix E explains two error conditions, traps and exceptional system status, and how to recover from them.

Appendix F lists page zero and hardware reserved locations.

Appendix G contains a Hollerith-ASCII conversion table.

Appendix H contains the ASCII character set.

Appendix I explains advanced multitask programming features for users who want to extend their multitasking environments.

The index alphabetically lists the concepts and terms in this book and references the pages on which they appear.

Several lists and forms follow the index.

"DG Offices" lists all Data General facilities world-wide.

"How to Order Technical Publications" points to the agencies from which order forms and manuals can be obtained.

"Technical Publications Comment Form" invites you to assist DGC in improving future publications by evaluating this book.

"Users' Group Membership Form" brings DGC software users together, in group meetings and through various publications, to exchange ideas, applications, problems, and solutions.

Related Manuals

Prerequisite readings are described under "Reading Paths" in this Preface.

Additional manuals describing RDOS are listed below.

RDOS/DOS Text Editor (DGC No. 069-400016)

SUPEREDIT Text Editor (DGC No. 069-400017)

RDOS/DOS Assembly Language and Programming Utilities (DGC No. 069-400019)

RDOS/DOS Debugging Utilities (DGC NO. 069-400020)

RDOS/DOS Backup Utilities (DGC No. 069-400022)

RDOS/DOS Sort/Merge and Vertical Format Utilities (DGC No. 069-400021)

Conventions

We use these conventions for command formats in this manual:

COMMAND required [*optional*] ...

Where	Means
COMMAND	Enter the command (or its accepted abbreviation) as shown. Upper-case letters indicate the command mnemonic.
required	Enter some argument (such as a filename). Sometimes, we use: $required_1$ | $required_2$ You can choose between the arguments listed. Do not use the vertical bar; it merely separates the choices. Lower-case italic letters indicate an argument.
[optional]	Brackets mean that you have the option of entering the argument. (Command switches also appear in this format.) Do not include the brackets in your code; they only set off the choices.
...	Repeat the preceding entry or entries. The explanation will tell you exactly what to repeat.
. . .	The process has continued without incident, and you may now take the next action described.

Additionally, we use certain symbols in special ways:

Symbol	Means
⟨CR⟩	Press the RETURN key on your keyboard.
□	Include a space at this point. (We use this to clarify in some cases. Normally, you can see where to put spaces.)

All numbers are decimal unless otherwise indicated, for example, 35_8.

In examples of dialogue, we use:

THIS TYPEFACE TO SHOW YOUR ENTRY

and

THIS TYPEFACE FOR SYSTEM RESPONSES.

R is the RDOS/DOS Command Line Interpreter prompt.

Table of Contents

Preface

Reading Path **i**
Organization **i**
Related Manuals **ii**
Conventions **ii**

Chapter 1
Overview

Generating an RDOS System **1**
Communicating with RDOS **1**
Program Development **2**
Higher-Level Languages **2**
Assembly Language **2**
Main Memory Considerations **2**
Foreground/Background Programming **2**
Mapped Features **3**
Device Access **3**
RDOS Organization **3**
System Library and Source Files **6**

Chapter 2
Files and Directories

Definition of a File **7**
File Overview **7**
Reserved Device Names **7**
Disk Filenames **9**
File Attributes and Characteristics **9**
Disk File Characteristics **10**
File Transfer **10**
Disk File Block Organization **10**
Sequentially Organized Files **11**
Randomly Organized Files **12**
Contiguously Organized Files **13**
RDOS Disk Directories **13**
Initial Disk Block Assignments **14**
System Directory (SYS.DR) **14**
Master Directory **15**
User Directories **15**
Partitions and Subdirectories **15**
Initializing and Releasing User Directories **17**
Referencing Disk Files **18**
Link Entries **18**
File Access Example **20**
Directory Command Summary **23**
Magnetic Tape Files **23**
Nine and Seven Track Data Words **24**
Tape File I/O **25**
Free Form I/O **25**
Initializing and Releasing a Tape Drive **25**
Referencing Tape Files with File I/O **26**
Linking to Tape Files **27**
Multiplexors **27**
Line 64 Reads **28**
Line 64 Writes (ALM and ULM only) **29**
ULM Line Codes **29**
Multiple Channels **29**
Modem Support Under RDOS **29**
Multiplexor Error Messages **30**
ALMSPD.SR **30**

Chapter 3
Single-task Programming

Multiple and Single-task Environments **33**
System Task Calls **33**
Status On Return From System Calls **34**
I/O Channel Numbers **34**
Commonly Used Commands **35**
Device and Directory Commands **39**
File Maintenance Commands **45**
File Attribute Commands **51**

Link Commands **53**
Input/Output Commands **56**
Console I/O Commands **72**
Memory Allocation Commands **74**
Device Access Commands **76**
Clock and Calendar Commands **78**
Spooling Commands **80**
Keyboard Interrupts **82**
Defining Interrupt Routines **83**
Summary **88**

Chapter 4
Extending User Address Space

Program Swapping and Chaining **91**
User Overlays **97**
Protecting User Memory Under Mapped RDOS **102**
Virtual Overlays **105**
Window Mapping **107**
Defining a Window Map **107**
Performing a Remap **109**
Extended Direct Block I/O **112**
Extended Direct Block I/O Example **115**
Summary **116**

Chapter 5
Multitask Programming

Task Priorities **117**
Task Control Blocks **117**
Building Multitask Programs **119**
Conserving ZREL Space **119**
Task States **119**
TCB Queues **120**
Task Synchronization and communication **121**
User Status Table **121**
Task and System Calls **123**
Task Initiation **123**
Task Termination **124**
Task State Modification **127**
Inter-task Communication **129**
Locking a Process Via Transmit and Receive Commands **130**
User Overlay Management **131**
Enqueuing Tasks **135**
User/System Clock Commands **138**
Managing Tasks by ID Number **141**
Task/Operator Communications Calls **144**
Task/Operator Communications Module (OPCOM) **146**
OPCOM Command Syntax **147**
OPCOM Command Example **153**
Disabling and Enabling the Multitask Environment **154**
Disabling and Enabling the Task Scheduler **155**
Summary **157**

Chapter 6
Foreground and Background Programming

Overview **159**
Dual Programming in Mapped Systems **160**
Executing Dual Programs **160**
Checkpointing a Background Program **161**
Dual Programming in Unmapped Systems **161**
Building Foreground Programs **161**
Executing Dual Programs **162**
Foreground/Background System Calls **164**
Summary **170**

Chapter 7
Interrupts and Power Failures

Servicing User Interrupts **171**
Commands for Interrupt and Power Fail Routines **171**
Power Fail/Auto Restart Procedures **175**
Power-up Service for User Devices **176**
Summary **176**

Chapter 8
Multiple Processor Systems

Overview **177**
Interprocessor Buffer (IPB) Programming **178**
Interval Timer **178**
Dual Processor Program Communications **178**
IPB Example **178**
MCA Programming **180**
Data Transmissions **180**
Using CLI Commands on MCA Lines **180**
Transmitting Copies of Systems or Stand-alone Programs **181**

Multiprocessor System
Illustration **182**

Chapter 9
System Tuning

Overview **185**
System Stacks, Cells, and
Buffers **185**
System Stack Requirements **186**
System Cell Requirements **186**
System Buffer Requirements **187**
How Tuning Works **189**

Chapter 10 **193**
Running In LEF Mode

Appendices **195**

Appendix A **197**
RDOS System and Task Calls

Appendix B **213**
User Parameters

Appendix C
Real-time Programming Examples

TIMEC Program **231**
EXAMPLE Program **234**

Appendix D **241**
Overlay Directory Structure

Appendix E
Exceptional System Status

Traps **243**
Exceptional Status **243**
Controlling Exceptional Status **244**
Producing a Core Dump **244**

Appendix F **249**
Page Zero and Hardware Reserved Locations

Appendix G **251**
Hollerith–ASCII Conversion Table

Appendix H **255**
ASCII Character Set

Appendix I
Advanced Multitask Programming

Definitions **257**
General Terms **257**
State Definitions **258**
Coding Your Own Task Calls **258**
TCB and Status Bits **258**
Scheduler Calls **258**
Handling Additional Task
Resources **262**
Task Scheduler Call-Outs **262**
Additional Resource Handler **265**
Operator Communications **266**
Task Control Block Values **266**

Index **269**

DG Offices

How to Order Technical Publications

ISD User Documentation Remarks Form

Users' Groups Membership Form

Figures

1.1 RDOS address space **5**
2.1 Sequential file block organization **11**
2.2 Random file block organization **12**
2.3 Contiguous file block organization **13**
2.4 Apportioning disk space **16**
2.5 Link entries **19**
2.6 Sample organization of an RDOS disk **21**
2.7 Data encoding (nine-track units) **24**
2.8 Data encoding (seven-track units) **24**
2.9 Data block structure **25**
2.10 Writing the first tape file **26**
2.11 Overwriting tape files **27**
3.1 Double-precision byte pointer **62**
3.2 Image binary code reading **65**
3.3 MTDIO status word bits **71**
3.4 Unmapped background memory **74**
3.5 Program with interrupt handler **83**
3.6 Program with .INTAD task **84**
3.7 Program interruption logic sequence **85**
4.1 Program swapping **93**
4.2 Program chaining **94**
4.3 User overlays **98**
4.4 Segment 1 of overlay file R0.OL **99**
4.5 Loading the overlay root programs **99**
4.6 Write-protecting memory **103**
4.7 Virtual overlays before .OVLD **106**
4.8 Virtual overlays after .OVLD **106**
4.9 Defining a window map **108**
4.10 Memory before remap **109**
4.11 Remapping **109**
4.12 Extended block read **115**
5.1 Task state/priority information (TPRST) **118**
5.2 TCB chain **120**
5.3 TOVLD logic sequence **133**
5.4 QTSK example **137**
5.5 Sample console commands and messages **153**
6.1 Loading foreground and background programs in an unmapped system **163**
8.1 Multiple processor line connections **180**
8.2 Multiprocessor system illustration **183**
9.1 Adequate cell apportionment **187**
9.2 Inadequate cell apportionment **187**
9.3 Disk blocks of the tuning file **190**
9.4 Details of the tuning summary report, first disk block **190**
9.5 Tuning overlay report **190**
B.1 PARU.LS **213**
C.1 TIMEC and TASK messages **231**
C.2 TIMEC flowchart **232**
C.3 TIMEC program listing **233**
C.4 EXAMPLE flowchart **235**
C.5 EXAMPLE program listing **236**
D.1 Overlay directory structure (multitask) **241**
E.1 Sample line printer dump **246**
F.1 Listing of PARS, giving page zero and hardware reserved locations **249**

Tables

1.1 System file names **6**
2.1 Reserved device names **8**
2.2 Initial disk block assignments **14**
2.3 Directory command summary **23**
2.4 Characteristic bits that affect multiplexors **28**
2.5 Selecting a ULM line speed **29**
2.6 Multiplexor error messages **30**
3.1 Commonly used commands **36**
3.2 Calls that control memory, returns and overlays **37**
3.3 Possible errors from calls that control memory, returns and overlays **38**
3.4 UFD template with displacement mnemonics **49**
3.5 Bit-attribute relationships **51**
3.6 Disk file characteristics assigned by RDOS **52**
3.7 Bits and associated device characteristics **53**
3.8 .MTDIO values returned **71**
3.9 System call summary **88**
4.1 System and task call summary **116**
5.1 Structure of a task control block (TCB) **118**
5.2 Structure of user status table (UST) **121**
5.3 User task queue table **136**
5.4 System, task, and OPCOM command summary **157**
6.1 System call summary **170**
7.1 System and task call summary **176**
9.1 System overlays and their functions **188**
A.1 RDOS command summary **197**
A.2 Error summary **210**
I.1 TCB words and how they can be changed **267**

Chapter 1

Overview

Data General's Real-Time Disk Operating System (RDOS) combines the advantages of a disk operating system with the speed of a memory-resident system. RDOS is real-time oriented: it can allocate program control to many tasks within separate foreground and background programs, while offering maximum efficiency and economy to a wide variety of installations.

Some of the major features that RDOS offers include:

- Disk and memory-residence
- Support for real-time FORTRAN IV, FORTRAN 5, DG/L, Extended and Business BASIC, and other advanced languages
- Support for BATCH processing
- A flexible file structure that allows disk partitioning and sharing of user files, buffered and unbuffered I/O and multiple user overlays
- Modular multitask levels of task priority
- 256 software levels of task priority
- Hardware mapping support for foreground/background programming, including protection and management of each program; access to mapped extended memory; and checkpointing of background programs
- Spooling (disk buffering) of output to slow peripherals
- Dual processor—shared disk support
- Multiprocessor support
- Tuning for improved performance

These and other basic RDOS concepts are explained in *Introduction to RDOS* (DGC No. 069-400010).

The minimum of hardware needed to run RDOS is a suitable Data General computer, a hard-copy or CRT terminal, and a disk. Larger versions of RDOS can support a real-time clock, power fail—auto restart, up to 16 megabytes of fixed-head disk storage, and more than 1,500 megabytes of moving-head disk storage. In addition, RDOS can support 16 magnetic tape units, multiple line printers, terminals, plotters, readers and punches, multiplexors, and CPUs. Mapped RDOS features hardware memory protection, and can support up to 256K bytes (NOVA) or 2M bytes (ECLIPSE) of memory.

Generating an RDOS System

Each system installation is unique; it must perform diverse tasks with one of many possible hardware combinations. An RDOS system is tailored to the user's environment with the system generation program, SYSGEN.

The builder of tailored operating systems, SYSGEN is an executable system program that can operate in any installation. Data General delivers a standardized starter (bootstrap) system with RDOS; this starter system, along with the SYSGEN program, enables the user to generate one or more configured systems. If you know your future requirements, you can generate other RDOS systems at this time to fulfill them. *How to Load and Generate RDOS* (DGC No. 069-400013) describes the procedures for doing so. The tailored system is bootstrapped into execution via BOOT, the RDOS bootstrap program.

Communicating with RDOS

You can communicate with RDOS and make it work for you in four ways:

via system and task calls in an assembly language program,

with Command Line Interpreter commands,

with the Batch monitor, or

indirectly, through a higher-level language.

The user writes system and task calls as instructions in a program, using the CLI as a dynamic console interface to RDOS. System and task calls activate logic within the system or task processing modules. Only those task-processing modules that the program needs become part of it.

The Command Line Interpreter (CLI) is a system utility program that accepts command lines from the console and translates them into commands to RDOS. Thus, the CLI is an interface between your console and the system. Unless

otherwise directed, RDOS will load the CLI at system initialization. RDOS will reload the CLI upon termination of a user program if the user did not chain to that program from the CLI (with the CLI CHAIN command). The CLI indicates that it is ready for input by outputting a ready message prompt, R, and a carriage return, and is interrupted when you press the keys CTRL and A, CTRL and C, or CTRL and F. (Keyboard interrupts are discussed further in Chapter 3.)

CLI commands allow the user to load programs, invoke other utility programs, and activate the BATCH monitor. BATCH executes jobs serially, without operator intervention, using job control commands in the job stream.

Advanced Data General compilers, and the BASIC interpreter, allow users to write programs in languages like DG/L, FORTRAN, and BASIC.

Program Development

Along with the CLI, Data General supplies a number of utility programs with RDOS. Each program is described in a separate manual, as listed in the Preface. The utilities help the user write code and develope it into useful, executable programs. During system generation, the utility programs are transferred to disk, making each of them accessible by a CLI command.

Your first step in program development is to write a source program that performs useful work for your computer application. This program can be written in a higher-level language like DG/L or FORTRAN, or in assembly language via one of the text editor utilities. The CLI's EDIT command invokes the Text Editor; Its MEDIT command invokes the Multiuser Text Editor; and its NSPEED or SPEED commands invoke the Supereditor. Your next step depends on whether you have used a higher-level language like FORTRAN, or assembly language. This manual will be most useful to assembly language requirements.

Higher-Level Languages

If you have written your program in FORTRAN, DG/L, or another higher-level language, you will compile and assemble it by invoking the appropriate utility with a CLI command. You will then use the Relocatable Loader utility, invoked with the CLI's RLDR command, to produce an executable program file. A program written with the BASIC interpreter can be corrected with the aid of the appropriate manual for that language, while using the CLI to access, maintain, and protect files and devices.

Assembly Language

A source program written in assembly language with the Text Editor or Supereditor utilities must be assembled into a relocatable binary file, using the CLI's ASM or MAC commands. After assembling the source program into a binary file, you will use another utility to process the binary file into an executable program, or save, file. This utility is the Relocatable Loader, invoked with the CLI's RLDR command. A program generally requires debugging the first time it is loaded; you may therefore load it with a symbolic debugger. You can then try to execute the program and, if it fails to run properly, debug it via the CLI's DEB command. CLI commands can be issued after any of these steps to maintain, protect, and examine the file.

Main Memory Considerations

Your computer arrived with a given amount of memory. The amount of this memory available for user programs will necessarily be a percentage of the total figure, as determined by the requirements of your tailored RDOS system. Each of the peripherals and software structures specified during system generation requires a certain portion of memory, as listed in *How to Load and Generate RDOS*. After deducting the system's memory from your maximum figure, you must also consider the space, aside from your own code, that each user program will actually require.

When you load a program, RLDR builds certain required tables, modules, directories, and the Task Scheduler into it. The code for each task call used is taken from the system library and loaded into the program. (Because system calls are executed in RDOS space, they require a minimum of user space.) These components require user memory space when the program executes. You may therefore want to conserve space by coding certain segments of the program as overlays. *Overlays* are called into memory one by one, as the program needs them; otherwise, they reside on disk. You define overlays within a program in the RLDR command line. Another way to extend effective user address space is to instruct an executing program to swap itself to disk, call a new program into memory, and return to memory when the new program has executed. This method, called *swapping*, has a variation called *chaining*. Overlays, swaps, and chains are described in Chapter 4, along with the extended memory available to users with mapped machines.

Foreground/Background Programming

You may want to run two logically distinct programs concurrently. RDOS allows the user to divide memory into two areas, called *foreground* and *background*, and to run a program simultaneously in each. When bootstrapped, RDOS starts up in the background; similarly, all executing programs run in the background until you command RDOS to execute one in the foreground. When running in two grounds, programs share such system resources as CPU time and I/O devices. The foreground program has priority unless otherwise specified by the user. Foreground and background programs can communicate with one another via system

calls or commonly-known disk files, as explained in Chapter 6.

A system that has no hardware mapping device is unmapped, and runs under unmapped RDOS. Such a system requires that memory be manually assigned to a program that runs in the foreground. You do this in the RLDR command line by specifying two starting addresses for the foreground program. They are the start of page zero relocatable memory, called ZREL, and the start of normal relocatable memory, called NREL. Once the program has been loaded and executed in the foreground, these addresses separate the two grounds. Up to 32K words of user address space, excluding RDOS space, can be directly addressed in an unmapped system.

Certain system calls, features, and CLI commands apply only to mapped systems. These exceptions are noted in the text. If a discussion makes no reference to mapping or the MAP unit, it applies to both mapped and unmapped systems.

Mapped Features

If your hardware features a MAP unit, it runs under mapped RDOS. In mapped RDOS, background and foreground programs can operate autonomously, either alone or via a CLI. Using mapped address space, both programs can share all memory not used by the system. Naturally, this amount depends on the total memory afforded by your computer and the size of your RDOS system, as determined by the features selected during system generation. Tools for accessing extended memory include virtual overlays and window mapping, as explained in Chapter 4. Any Data General computer with mapping hardware can support mapped RDOS.

Addresses are specified similarly in mapped and unmapped systems, except that a mapped system can remap addresses in pages of 1,024 words. Addresses in mapped systems are called *logical*, instead of *physical*, addresses.

When you run two programs, the system maps them separately; each program is aware of its address space only, and cannot reference locations outside it. The system allots memory to each program according to its highest address. It assigns each program a complete logical address space from page zero through its highest address NMAX, in 1,024-word pages.

When RDOS starts up, it assigns all memory to the background; you reserve memory for the foreground with the CLI's SMEM command, and execute a program in the foreground with the EXFG command.

Aside from hardware separation of foreground and background address space, the mapped system protects itself in three ways: it guards system devices, prevents infinite address defers, and protects data channel operations.

Device Access

Initially, no user can access any device directly, including the MAP and CPU, on a machine-language level. If the user attempts to reference a device on a machine level without having been enabled to do so, the system refuses the request, prints a "trap" message, creates a break save file called BREAK.SV, and returns to a higher-level program—usually the CLI. The system responds in the same way if it encounters more than 16 levels of indirect address—that is, it traps, creates the break file, and returns. Appendix E describes traps in more detail.

Users can gain direct access to any system device—and avoid the map's safeguards—by using the system call .DEBL, discussed in Chapter 3. The map also monitors the data channel and allows user devices to access it through the system call .STMAP, described in Chapter 7. You can include your own devices in a system and allow them to communicate with RDOS by writing device drivers and user interrupt service routines for them.

RDOS Organization

The RDOS executive is the main framework of the operating system and must be memory-resident before any processing can occur. This resident portion of RDOS processes system calls and interrupts, and manages RDOS buffers. Other modules of the system reside in system overlays; they are brought into memory from disk as required to perform such functions as initializing the system, opening, closing, renaming or deleting files, and spooling control.

In an unmapped system, the RDOS executive resides at the top and bottom of memory. Locations 0 through 15_8 contain program and interrupt entry points into the top area of RDOS. In a mapped system, resident RDOS begins at location 0 and extends to the highest address required; it is invisible to user programs. Above resident RDOS in all systems—and at the very top of memory in unmapped systems—is a series of system buffers. The system buffers handle buffered I/O transfers, and hold system overlays and directories from disk.

The portion of page zero memory available for user programs begins at location 16_8 (labelled USP); skips to locations 20_8 through 37_8; and then extends from 50_8 through address 377_8. In an unmapped system, these are physical addreses; in a mapped system, they are logical addresses. NREL memory is allocated in much the same way for both mapped and unmapped systems. In a mapped system, ZREL and NREL addresses are logical; in an unmapped system, they are absolute. This distinction is not important, however, to user programs.

Above program ZREL, the Relocatable Loader (RLDR) builds a User Status Table, called UST, for your program.

This table starts at address 400_8 in an unmapped background, and at logical address 400_8 in both the mapped foreground and mapped background areas. The UST describes, among other things, your program's length, number of tasks required, and number of I/O channels needed.

Above the UST RDOS reserves an area for a pool of Task Control Blocks (TCBs). RDOS uses TCBs to store task state information, such as the state of the accumulators and carry. If you have defined overlays in your program via RLDR, an overlay directory resides above the TCBs. And above the overlay directory, if any, is NREL memory, which holds the rest of your program. RLDR reserves a node, or vacant space, in the program for each overlay segment you defined; overlays from each group will occupy this mode one by one.

Above your background or foreground program, but still in NREL memory, are the task-processing modules and Task Scheduler that it requires in order to run. RLDR searches the system library for these components and places them on disk with your program. During execution, they are generally highest in NREL memory.

Figure 1.1 is a simplified illustration of unmapped and mapped memory, in which each system is running foreground and background programs, and each program has one overlay node.

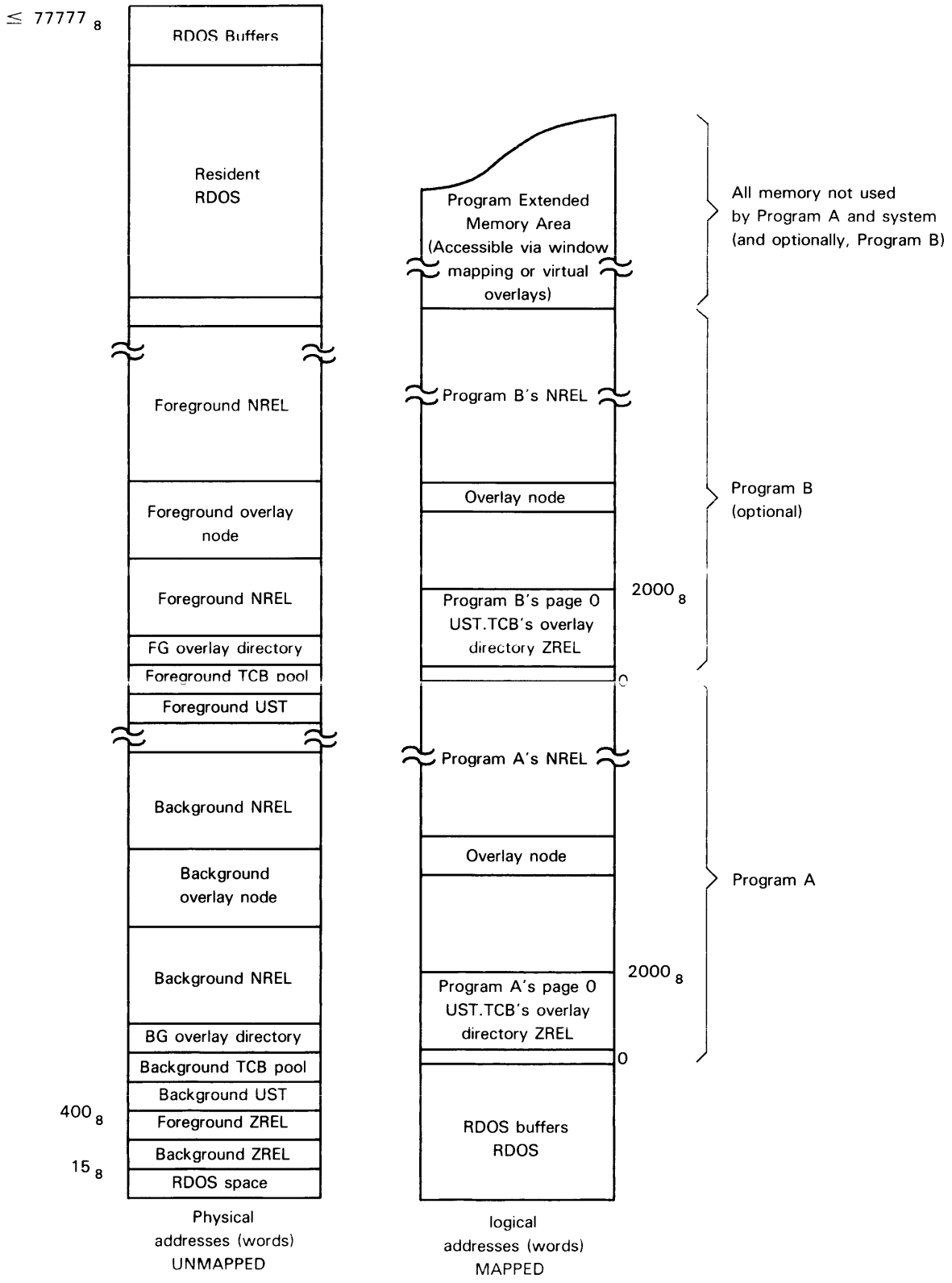


Figure 1.1 RDOS address space

ID-00334

System Library and Source Files

The system library, named SYS.LB, contains task-processing modules, task schedulers, and other useful routines for user programs.

Other files supplied with your system contain definitions for system features and for system and user parameters. Depending on the programs you write, you may want to include some or all of these files in the Macroassembler's permanent symbol file, MAC.PS, as described in *RDOS/DOS Assembly Language and Programming Utilities* (DGC No. 069-400019). The CLI's LIST or PRINT commands can be used to display the files' contents or obtain a hard copy. Table 1.1 lists and describes the most common of these files.

Filename	Description	Where Used
PARU.SR	User parameter file containing mnemonics for all system constants and errors (see Appendix B for a listing of PARU.SR)	Aids in assembly language programming on all Data General computers
PARS.SR	System parameter file containing internal RDOS constants and some macros for system-level tables, such as device control blocks and certain buffers	All Data General computers
NBID.SR	NOVA basic instruction definition, providing the basic instruction set for all DG machines	All Data General computers
OSID.SR	Operating system instruction definition	All Data General computers
(varies)	Multiply-divide instructions, provided with your language	All Data General computers
LITMACS.SR	Literal macros, used by RDOS, which can also be incorporated in the user's programs	All Data General computers
NFPID.SR	Floating point instructions	Computers with floating point hardware
FPID.SR	Floating point instructions	Computers with floating point software (floating point interpreter package)
ALMSPD.SR	Contains default characteristics of multiplexed lines. Can be edited, assembled, and included in tailored system to reflect special line configurations (see Chapter 2 for details)	All Data General computers
NSID.SR	Stack instruction definition	NOVA 3 and NOVA 4 computers
RDOS.SR	Unmapped RDOS switch settings	All NOVA computers
NRDOS.SR	Mapped RDOS	NOVA 3 and NOVA 4 computers
MRDOS.SR	Mapped RDOS switch settings	All mapped NOVA computers except 830s and 840s
NEID.SR	NOVA extended instruction definitions	ECLIPSE computers
NCID.SR	Commercial ECLIPSE instructions	ECLIPSE computers
BRDOS.SR	Unmapped RDOS switch settings	ECLIPSE computers
ARDOS.SR	Mapped RDOS switch settings	Mapped S/200 and C/300 ECLIPSE computers
ZRDOS.SR	Mapped RDOS switch settings	Mapped ECLIPSE computers except S/200 and C/300
TRDOS.SR	Mapped RDOS system instructions	ECLIPSE S/20 computer

Table 1.1 System file names

Chapter 2

Files and Directories

This chapter defines the different RDOS media for files—generally disk and magnetic tape—and explains how to use each medium. A section on disk files describes the mechanisms used to organize and speed up access to files on disk, and outlines the file structure that RDOS imposes on every disk it uses. These mechanisms include directories—called partitions and subdirectories—which contain groups of files, and link entries. Link entries allow users in different directories to use a single file. The chapter concludes with a description of multiplexors.

Definition of a File

A file is any collection of information, or one of several devices for receiving or sending that information. Typical examples of both file types include:

source files
relocatable binary files
executable program files (save files)
listing files
teletypewriter or CRT keyboards
teletypewriter printers or CRT screens
line printers
magnetic tape files

Source, binary, program, and listing files have special characteristics; each represents a step in program development. The developer writes a source file with a text editor and inputs it to an assembler, which produces a relocatable binary file. The relocatable binary file is processed with the loader; as a result, the file is placed on disk, with absolute location data, as a save file. A save file is an executable program version of the original. Each save file is a core-image file: it is stored on disk word-for-word as it will be loaded into memory and executed.

Unless otherwise specified, the keyboard and printer or screen are the default input and output files for most system operations. The line printer is another type of output file. Magnetic tape files are discussed extensively later in this chapter. Cassette files are handled exactly as magnetic tape.

File Overview

All devices and disk files are accessed by filename; all magnetic tape files are accessed by device name and file number. Both reserved device names and disk filenames are discussed at the outset of this section.

A file must be opened—that is, associated with an RDOS channel via an .OPEN system call—before the user can access it. The CLI's file I/O commands do this automatically, but an RDOS program must be coded to open any files that it needs. You can open a disk file and allow several concurrent users to access and modify its contents; you might open it exclusively, permitting only one user to modify the file and allowing others to read it; or you could open it for reading only by several users. This section describes the attributes that control file access in general terms.

Finally in this section, certain characteristics of disk files and methods of transferring one file to another file or device are briefly discussed.

Reserved Device Names

I/O devices have special names, most of them beginning with the character $. Within the limits of the particular device, each device name can be used in a command exactly as a disk file's name would be. Table 2.1 shows how to enter each device name reserved by RDOS.

Device Name	Device
$CDR	Punched card reader; mark sense card reader.
CT*n*	Data General cassette unit *n*, first controller, where *n* is in the range of 0–7.
DK0	Data General model 6001-6008 fixed-head disk, first controller.
DP*n*	Data General moving-head disk pack, first controller, where *n* is a unit numbered 0, 1, 2, or 3, second controller where *n* is a unit numbered 4, 5, 6, or 7.
DP*n*F	Top loader (dual-platter Disk Subsystem), first controller, where *n* is a unit numbered 0, 1, 2, or 3, second controller where *n* is a unit numbered 4, 5, 6, or 7. Each unit has two disks. The top (removable) disk is DP*n*, the fixed disk DP*n*F. This controller also supports diskette drives.
DS*n*	Data General Model 6063/6064 fixed-head disk. The 6063 is single-density, the 6064 is double-density, and *n* is a unit numbered 0, 1, 2, or 3.
DZ*n*	6060-series and 6122, 6160, 6161 disk units, first controller, where *n* is 0, 1, 2, or 3, second controller where *n* is a unit numbered 4, 5, 6, or 7. Model 6060 uses single-density disks; 6061 uses double-density disks.
$DPI	Input dual processor link (see Chapter 8).
$DPO	Output dual processor link (see Chapter 8).
$LPT	80- or 132-column line printer.
$LPT1	Second line printer.
MCAR	Multiprocessor communications adapter receiver.
MCAT	Multiprocessor communications adapter transmitter.
MT*n*	7- or 9-track magnetic tape transport, first controller, where *n* is in the range of 0–7, second controller where *n* is in the range of 10–17.
$PLT	Incremental plotter.
$PLT1	Second incremental plotter.
$PTP	High-speed paper tape punch.
$PTP1	Second paper tape punch.
$PTR	High-speed paper tape reader.
$PTR1	Second paper tape reader.

Table 2.1 Reserved device names

Device Name	Device
QTY	Asynchronous line multiplexor (ALM), asynchronous data communications multiplexor (QTY), or Universal line multiplexor (ULM).
$TTI	Teletypewriter or display terminal keyboard*.
$TTI1	Second teletypewriter or display terminal keyboard.
$TTO	Teletypewriter printer or CRT display.
$TTO1	Second teletypewriter printer or CRT display.
$TTP	Teletypewriter punch.
$TTP1	Second teletypewriter punch.
$TTR	Teletypewriter reader.
$TTR1	Second teletypewriter reader.

Table 2.1 Reserved device names (continued)
*For most devices, RDOS supplies an end-of-file mark. On $TTI and QTY input, however, you must indicate an end-of-file by pressing the CTRL and Z keys (CTRL-Z).

Aside from the ALM and QTY, device drivers have been written reentrantly, allowing RDOS to support devices in pairs. Thus, an RDOS system can support two controllers for every type of disk drive that Data General provides. Each controller—except for models 6001-6008—can supervise up to four disk drives. The 6045 or 4234 controllers can support both disk and diskette drives. Use the following names to address second device controllers on your system:

DK1	Second Data General fixed-head disk.
DP*n*	Second moving-head disk pack controller, where *n* is 4, 5, 6, or 7.
DP*n*F	Second top loader (model 6045 or 4234A) controller, where *n* is 4, 5, 6, or 7. The removeable disk is DP*n*, and the fixed disk is DP*n*F.
DS*n*	Second 6063/6064 fixed-head disk controller, where *n* is 4, 5, 6, or 7.
DZ*n*	6060-series unit, second controller, where *n* is 4, 5, 6, or 7.
CT*n*	Second cassette controller, where *n* is 10 through 17 octal.
MT*n*	Second magnetic tape controller, where *n* is 10 through 17 octal.

For other secondary device names, append a 1 to the primary name, for example, $LPT1, $PTP1, $CDR1, and so on.

Disk Filenames

A disk filename is a string of up to 10 ASCII characters, including upper- and lower-case letters, numbers, and the dollar sign ($).(RDOS converts lower-case letters to upper-case by default.) The string is packed from left to right and terminated by a carriage return, form feed, space, or null. A filename may consist of any number of characters, but the system recognizes only the first ten. The dollar sign can also be used freely in a disk filename. However, the reserved device name combinations should be avoided.

A disk filename may contain an extension—a period followed by one or two alphanumeric characters, which may include the dollar sign. Although an extension may consist of any number of characters, the system recognizes only the first two. SIMULATOR.SV is an example of a filename with an extension.

The CLI often appends an extension to a filename to indicate the type of information the file contains and to distinguish it from other types of files created from the same source file. Assume, for example, that your source file is named FORECAST.SR. The CLI will append extensions to different versions of FORECAST as follows:

FORECAST.RB	Relocatable binary file (after assembling source file).
FORECAST.SV	Core image, or save, file (after loading or binding binary file).
FORECAST.LS	Listing file (only if such a file was specified during the assembly step).
FORECAST.OL	Overlay file (only if overlays were specified in the load or bind command).

While developing source programs into executable save files via the system assemblers and binders, you may ignore extensions if you assign the extension .SR, or no extension, to your assembly language source files. The utilities use a search algorithm to find the file with the appropriate extension. RDOS will always be given the extension .SV. Although RDOS assigns the extension .SV to each executable program, you need not enter .SV to execute it. Instead, simply type the file's name—for example, FORECAST ⟨CR⟩—from your console. If you append a unique extension to a filename, you must always include it when accessing the file via the CLI or a system call. (Save files will not execute with an extension other than .SV.) When adding your own extension to a filename, either avoid a CLI extension or use it properly. Do not, for example, confuse the CLI by giving a source file the extension .SV.

File Attributes and Characteristics

A file's attributes protect it by permitting or restricting the functions of reading, writing, renaming, deleting, or linking.

The attributes listed here apply primarily to disk files. RDOS protects nondisk files by assigning attributes that cannot be altered. (Of course, the user can write-protect a file on magnetic tape by removing the write-enable ring.) Use either the RDOS system call .CHATR (Chapter 3) or the CLI command CHATR to change the access attributes of a file.

P	Permanent file. No user can delete or rename a file that has this attribute.
S	Save file (core image). RLDR assigns this attribute automatically, and no file can be executed without it.
W	Write-protected file, which no one can modify.
R	Read-protected file, which no one can read.
A	Attribute-protected file, whose attributes cannot be changed. Once the A attribute has been set, it cannot be removed.
N	No resolution permitted. A file with this attribute cannot be linked to.
?	First user-definable attribute. When placed in bit 9 of the attributes word, permits the user to assign his own file attribute. (See Chapter 3, under the .CHATR command, for details.)
&	Second user-definable attribute. When placed in bit 10 of the attributes word, permits the user to assign his own file attribute. (See Chapter 3, under the .CHATR command, for details.)

Note that user-defined attributes should not be more restrictive than the file requires. A file with the attributes AP, for example, can only be deleted by erasing the entire disk with a procedure called full initialization.

Disk File Characteristics

Disk file characteristics are determined when you create a disk file, and cannot be changed thereafter. These characteristics include:

D Randomly organized file. (All save files must have this characteristic.)

C Contiguously organized file.

L Link entry. (Such a file contains nothing, but points to another file.)

T Partition file. (All partitions also have the C characteristic.)

Y Directory file. (A directory may include partitions and subdirectories.)

The CLI's LIST/A command allows the user to obtain information from a file directory about one or more files.

File Transfer

The CLI's XFER command copies a file from one device to any other. It requires two arguments:

```
XFER sourcefile destinationfile
```

For example, the statement

```
XFER @MT0:0 INDEX ⟨CR⟩
```

causes the CLI to create a disk file named INDEX and to copy to it the contents of the first file on magnetic tape unit 0. (The symbol ⟨CR⟩ represents a carriage return.) In a second example, this statement

```
XFER MYFILE YOURFILE ⟨CR⟩
```

creates YOURFILE on disk and transfers the contents of MYFILE to it.

Disk File Block Organization

The primary unit in an RDOS disk file is the disk block, which contains 256 16-bit words, or 512 bytes. When you create a disk file, the system call or CLI command directs the system to organize the file in one of three ways: sequentially, randomly, or contiguously.

In a *sequential file*, the system reads disk blocks in logical sequence, one by one. It reserves the last word, or last two words (depending on the disk), for a pointer to the next block. RDOS always reads and writes sequential files in blocks via system buffers, which slows the process significantly. Sequential files are created with the system call .CREAT or the CLI command CREATE.

In a *random file*, the system uses a file index to access any block. Generally, no more than two disk accesses are needed to access a block. (Very large files may require more.) RDOS uses all 256 words for data storage. Random file blocks can be read or written via direct block I/O, without system buffering, to save time. To create a random file, use the system call .CRAND or CLI command CRAND.

In a *contiguous file*, access is the fastest because all blocks are contiguous on disk and each contiguous file has a fixed, unalterable length in blocks. This means that RDOS does not need a file index and requires only one disk access. Each block uses all 256 words for data storage. Direct block I/O can be used for contiguous files. They are created with the system calls .CCONT or .CONN, or with the CLI command CCONT. Chapter 3 explains the difference between .CCONT and .CONN.

RDOS offers five ways to access disk files for I/O. In all but direct block I/O, RDOS transfers files via system buffers. Chapter 3 discusses the I/O modes in detail.

Sequentially Organized Files

When the system writes a sequential file to disk, the first block has relative number 0, the second 1, and so on. RDOS assigns each block a logical address, which it uses to derive the block's physical sector/track location on disk. In the last word of this block (or last two words on multiple-platter disks), RDOS stores a link to the next block. This link is invisible to the user but not to RDOS, which uses it to compute the physical address of the next relative block.

Assume, for example, that RDOS is reading block 0 of a sequential file. When it reaches the link at the end, RDOS finds the logical address of block 1, moves to block 1, and continues reading. Blocks 0 and 1 need not be contiguous on disk. From block 1, RDOS reads forward but can never skip a block. Thus, to reach block 7, RDOS would have to read forward until it encountered the link at the end of block 6. Figure 2.1 illustrates this concept.

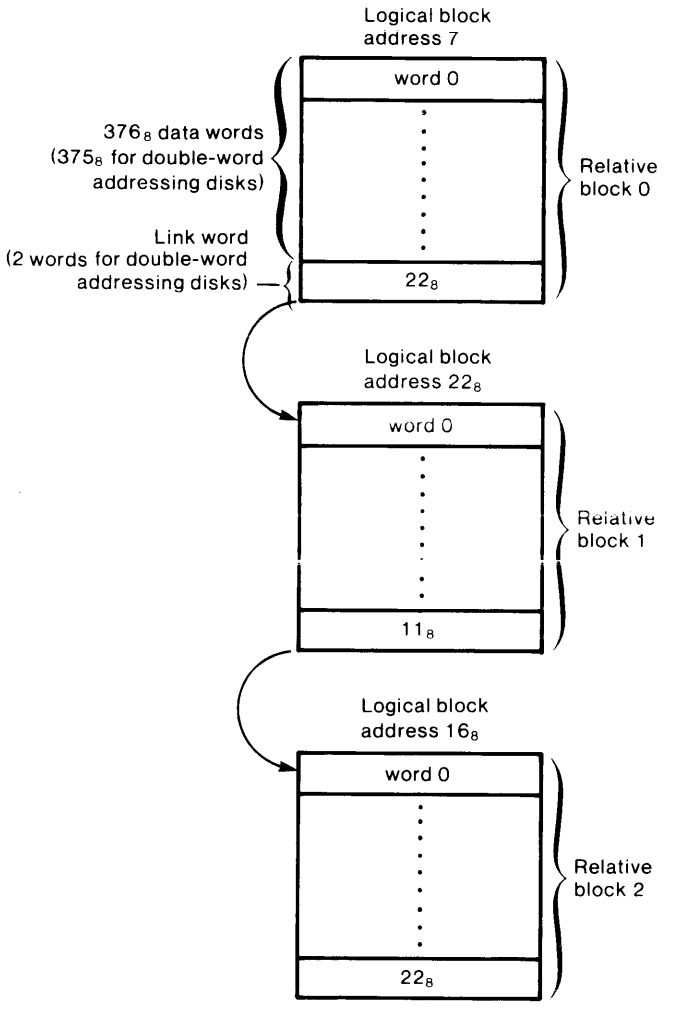


Figure 2.1 Sequential file block organization

SD-00534

When you access a sequential file for I/O, RDOS transfers it via system buffers. Block by block, RDOS reads the file into a system buffer for the transfer. When writing data into its system buffer area, RDOS overwrites the oldest available buffer block first. When all buffers have been used, the least-recently used is the first to be overwritten. After RDOS has read a block into its buffers, you can read or write the block's records directly; no further disk access is required.

Randomly Organized Files

In RDOS, all save files employ random organization. RDOS creates a file index for any random file that you create. In this index, the system enters one or two words, depending on the disk size, for each block that you write in the file. These index entries contain the block's logical disk address, allowing you to access any block on the disk. Index blocks are linked in the same way as sequential blocks, except that the last word or two points to the next index block. The first data block in the file is numbered 0, the second 1, and so on; the first entry in the index, entry 0, contains the logical address of block 0, and so on. If an index entry contains zeros, or no address, its corresponding block has not been written.

Figure 2.2 shows the relationship between the file index and data blocks in a randomly organized disk file.

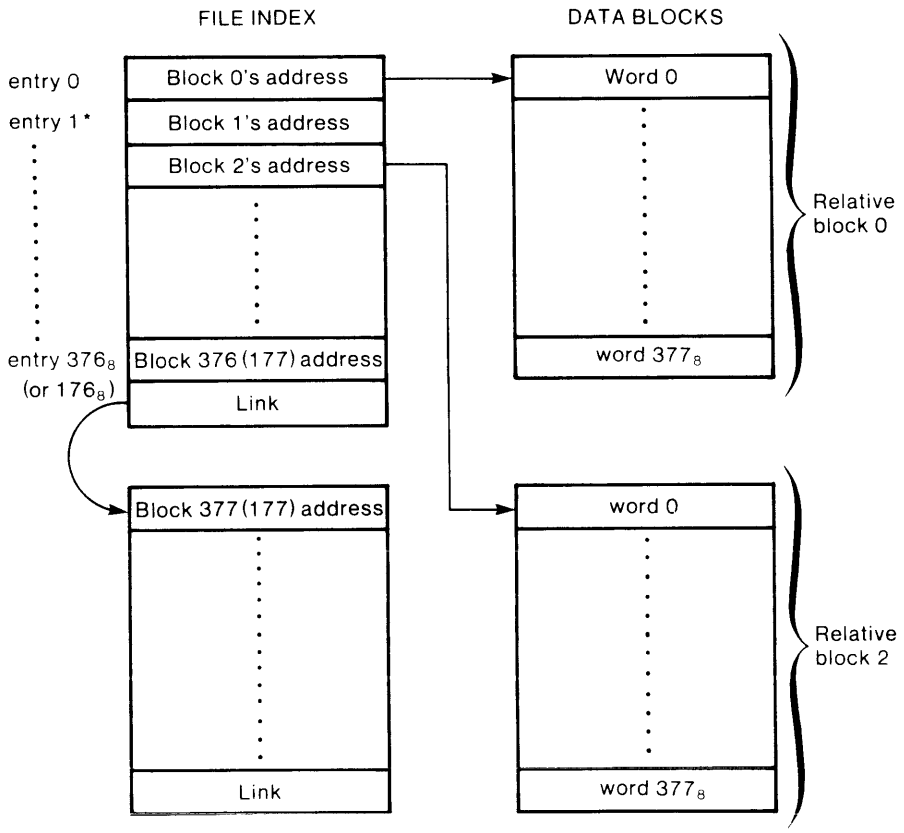


Figure 2.2 Random file block organization SD-00535

For files that contain less than 255 data blocks, RDOS generally needs only two disk accesses to read or write a block: one for the file index, and one for the block of data itself. If the file index is memory-resident—as it would be if you accessed the file previously and the index remained in a system buffer—only one access need be made. If the data block itself resides in memory, RDOS requires no disk accesses at all.

You can use all I/O commands available for sequential or random files. Because random organization is more efficient, I/O is generally faster on these files. For large-scale I/O, processing time can be shortened even further by using direct block I/O commands to transfer random files. Direct block I/O transfers cause RDOS to transfer an entire block from disk to the specified memory area without using system buffers. By avoiding buffering, you save time but forfeit the automatic management of the system buffers.

Contiguously Organized Files

As shown in Figure 2.3, RDOS accesses data blocks in contiguously organized files randomly, without a file index. Contiguous files consist of a fixed number of disk blocks located at an unbroken series of disk block addresses. The user can neither expand nor reduce the size of these files. Since the data blocks are at sequential logical block addresses, all that RDOS requires to access a block within a contiguous file is the address of its first block, or the file's name, and the relative block number within it. RDOS organizes all disk partitions and overlay files contiguously.

All I/O operations permitted on randomly organized files can be performed on contiguous ones, but the size of the contiguous file remains fixed. Block access is faster in a contiguous file, because RDOS does not need to read a file index.

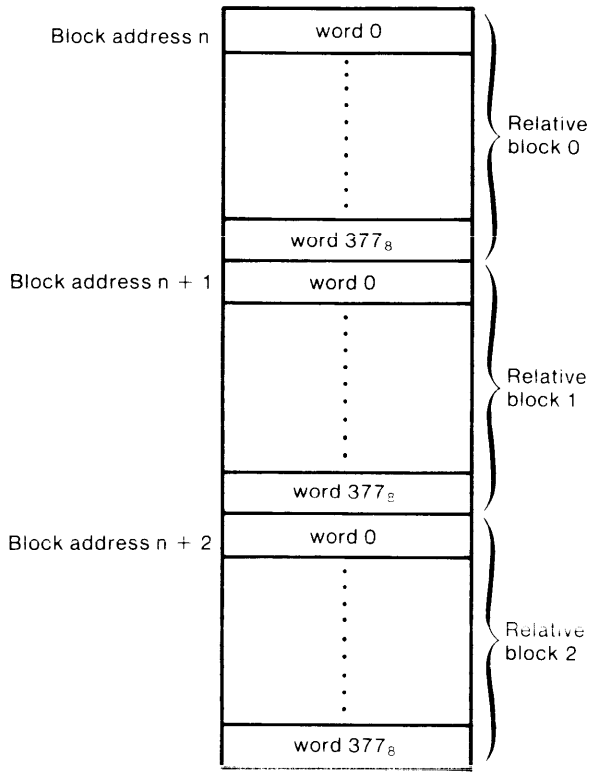


Figure 2.3 Contiguous file block organization SD-00536

RDOS Disk Directories

Before introducing a disk to the system, the user must check and fully initialize it with the disk initializer, DKINIT.SV. DKINIT, a stand-alone program, accompanies RDOS on your DG-supplied release tape or diskette, and is described in *How to Load and Generate RDOS*. After running DKINIT on the disk, the user may elect to install a bootstrap root on it. This routine enables you to bootstrap an RDOS system on any other disk from this one, as long as the new disk also contains the program BOOT.SV. The bootstrap root occupies blocks 0 and 1 of the disk; the disk ID is in block 3; and the bad block pool created by DKINIT occupies block 4.

When a disk is first introduced into the system, RDOS creates on it two system directories, SYS.DR and MAP.DR. SYS.DR records all filenames and other file data on the disk, and is updated by RDOS whenever you create, modify, or delete a file or user directory. MAP.DR is a block allocation map. It records those blocks which are in use and those that are free for data storage. MAP.DR is aware of all disk space except blocks 0 through 5.

Initial Disk Block Assignments

Certain blocks on every disk have fixed assignments, while the remaining blocks are free for system use or file storage. Table 2.2 shows the initial block assignments on an RDOS disk.

Disk block no. (octal)	Assignment
0,1	Root portion of the disk bootstrap program, BOOT
3	Disk ID
4,5	Bad block pool index
6	First index block of SYS.DR
7	Index of file index blocks used whenever a program swap occurs
8 to 16	Storage for swap file index blocks
17 to *n*	MAP.DR blocks, where *n* depends on disk size
(n+1) to *m*	BOOTSYS.OL (always allocated by INIT/F)
.	
.	
.	Free blocks for RDOS or user files

Table 2.2 Initial disk block assignments

The MAP.DR file starts at block 17_8. It is a contiguous file. Each bit of each word in MAP.DR indicates whether or not a specific block is in use, as follows:

Word (octal)	**Contents**
0	Block allocation map. One bit for each block, from left to right in ascending order, starting with block number 6. 0 means that a block is available, 1 means that a block is in use.
.	
.	
.	
n—1	Variable *n* represents the size of the partition in blocks divided by 16 (integer division).

System Directory (SYS.DR)

A user can create many directories within an RDOS system, and numerous files in each directory. RDOS creates a SYS.DR for each directory to keep track of the files within it. Each copy of SYS.DR is a random file.

The system directory employs a hashing algorithm to speed up access of directory entries. RDOS allocates an initial system directory area when the disk is initialized with DKINIT.SV. This area, called a frame, is a contiguous set of disk blocks, minimizing head travel time. Users can check and modify the frame size on a disk with DKINIT commands.

The first word in each block of SYS.DR is the number of files listed in the block. Following this word is a series of 22_8-word entries called *user file descriptors*, or UFDs, which describe each file. Each block in SYS.DR is composed as follows:

Word (octal)	**Contents**
0	Number of files in this block of the directory (16_8 maximum)
1	
.	
.	User file descriptor (UFD)
.	
22	
23	
.	
.	User file descriptor (UFD)
.	
44	
.	
.	Remainder of block
.	
376	Contains maximum number of UFDs that ever existed in this block; if the number is 16, an overflow block may exist.

The UFD describes the file's name, its two-character name extension, its size, its attributes and characteristics, the address of the first block, other qualities, and a logical code for the device that holds this file, as follows:

Word (octal)	**Contents**
0—4	Filename (padded with nulls, if necessary)
5	Extension (padded with nulls, if necessary)
6	Attributes and characteristics
7	Link access attributes
10	Number of last block in file
11	Byte count in last block
12	First address (physical address of first block in sequential or contiguous file, or first block of index for a random file)
13	Year and day last accessed
14	Year and day created or most recently modified
15	Hour and minute created or most recently modified
16	UFD variable information
17	UFD variable information
20	Use count
21	Device code DCT link

The attributes in words 6 and 7 permit or restrict access to the file, as explained in the discussion of .CHATR and .CHLAT in Chapter 3. A nonzero file use count indicates that one or more users have opened the file. If a malfunction occurs when a file is open, its count will often be incorrect, requiring that you clear it to zero (via the CLI's CLEAR command) before closing, renaming, or deleting the file.

Master Directory

The master directory (device) on each disk has the following uses:

- It becomes the current directory after you bring up the system, bootstrap a new system, or release a different current directory.
- It contains the current RDOS system save and overlay files, and usually contains the system utilities and library unless they were loaded into another directory, or were never loaded or copied.
- It contains push space for program swaps.
- It holds the spool files and tuning file, if any.

The master directory is determined when you bootstrap RDOS into operation. It remains the master until released, or until you bootstrap another system or program via the CLI's BOOT command.

User Directories

Within any RDOS system, each user requires disk space for files. Disk partitions and subdirectories permit you to organize and assign file space flexibly, by user or category name.

Although either CLI commands or system calls can be used to organize disk space, the CLI is the method of choice. Error interpretation is faster and simpler via the CLI. Once a hierarchy has been created from the console, you can access its directories and manipulate files via system calls in your programs.

Partitions and Subdirectories

Each disk introduced to the system contains a given number of blocks available for storage. These blocks comprise an area called the *primary partition*. Sections of the primary partition can be logically detached and assigned different filenames, according to the users' needs. These discrete sections are called *secondary partitions*; you create them and give them a fixed size with the CLI's CPART command or .CPAR system call.

Within the primary partition (and secondary partitions, if any) are smaller groups called *subdirectories*. You create a subdirectory with the CLI's CDIR command or .CDIR system call. Each subdirectory is flexible: it grows or shrinks according to the files that you append to or delete from it. A file may also exist in the master directory. A subdirectory and its files can never outgrow the fixed size of its parent partition. A newly-created subdirectory consists of three blocks: SYS.DR's initial index block, and two data blocks for the SYS.DR and MAP.DR entries.

In a multiuser RDOS system, the type of disk space a user receives depends on the installation. Typically, each user has a personal directory and unlimited reading access to several, common public files. In some systems, each user has a large secondary partition for subdirectories and files; in others, each has a subdirectory on the primary partition.

Figure 2.4 shows a disk before and after partitioning, along with the CLI commands that make partitioning possible. D*Xn* is a general term that varies according to your own disk name(s), as described in Table 2.1.

Each primary partition, secondary partition, and subdirectory contains a version of the disk's system directory to keep track of the files within it and enable it to access I/O devices. Each partition's SYS.DR also contains a version of the map directory to maintain a record of free and occupied data blocks. Each subdirectory's SYS.DR uses a copy of its parent partition's MAP.DR.

One important advantage of secondary partitions is that a disk failure in a secondary partition will not affect files in other partitions. This is because the map directories in other partitions are not vulnerable to a failure. For this reason, some users prefer to place their systems and utilities in a secondary partition, and operate from that partition, using directory specifiers.

Parititons are contiguous files, while subdirectories are randomly organized. Both are unusual in that they contain other files and receive the extension .DR, but are no more privileged than data files. You can dump, list, or load partitions and subdirectories, and delete all but the primary partition.

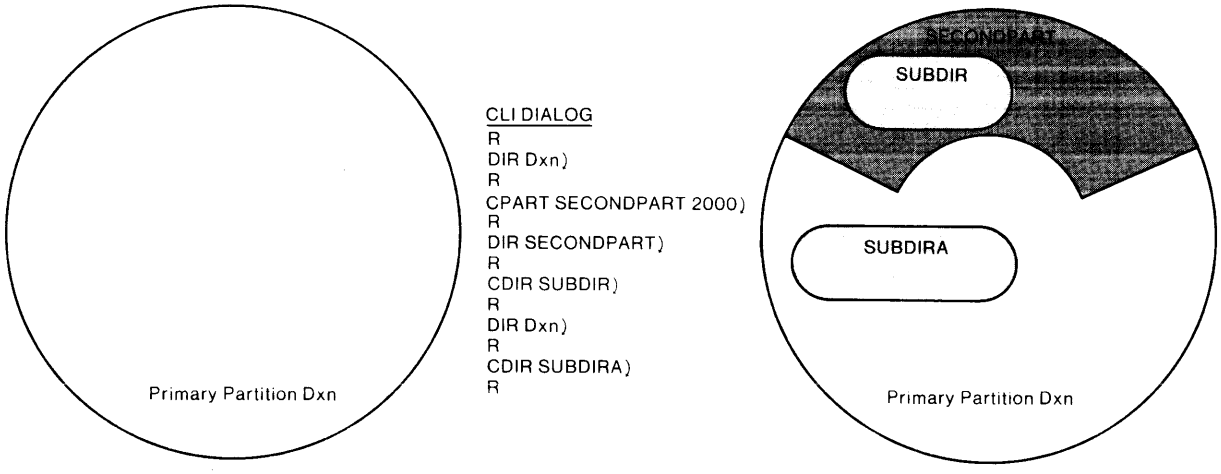


Figure 2.4 Apportioning disk space

SD-00537

Initializing and Releasing User Directories

Subdirectories and partitions must be initialized before you can access the files or subdirectories within them. Initialization opens a subdirectory or partition, introduces it to the system, and prepares it for use. This procedure is called *partial initialization*. (*Full initialization* introduces new disks to the operating system; it writes a new SYS.DR, MAP.DR, and BOOTSYS.OL on the disk, effectively destroying all existing file structures.)

Once you have bootstrapped RDOS and set the parameters of date and time, the CLI displays its R prompt. At this point RDOS has initialized only the master directory, which holds the current RDOS system. Most often the master directory is DP0, DP0F, DZ0, or DS0, but it may also be another disk or secondary partition.

The CLI's INIT or DIR commands (or the system calls .INIT or .DIR) are used to initialize a subdirectory or partition, for example:

```
R
INIT partition-or-subdirectory ⟨CR⟩
```

While many partitions and subdirectories can be initialized at any moment, RDOS allows only one current directory at a time. The current directory is the one in which RDOS searches for all files unless you have directed it to search elsewhere. The DIR command simultaneously selects and initializes a new current directory, for example:

```
R
DIR partition-or-subdirectory ⟨CR⟩
```

During system generation, the user specifies a maximum number of subdirectories and partitions that can be initialized at any moment. The current maximum is 64. If the number of initializations exceeds the maximum defined for your system, the CLI returns an error message (or the program takes an error return).

Once a directory has been initialized, it is part of the system; RDOS will remember where it is, and access it, even if it resides on another partition or subdirectory. It remains in the system until you release it. The CLI's RELEASE command (or system call .RLSE) performs this function, for example:

```
R
RELEASE subdirectory-or-partition ⟨CR⟩
```

The act of releasing a directory removes its initialization. When the current directory is released, the master directory becomes current until you specify another directory via DIR or .DIR. The master directory holds the operating system, which closes down when you release it.

During an orderly shutdown, the master directory is released via the CLI. This directory must be released before physically removing the disk that holds it. If two programs are running, the foreground program must be terminated and the master directory released from the background console. RDOS verifies the release as follows:

```
R
RELEASE DP0 ⟨CR⟩
MASTER DEVICE RELEASED
```

At this point you may turn off the computer, disk drive(s), and peripherals.

When more than one disk unit is present in the system, a *global directory specifier* is required to initialize each one. Global specifiers were listed earlier under "Reserved Device Names"; examples include DP0 and DP0F (removable and nonremovable disks in unit 0 of the first top-loader controller), and DZ0 (first 6060-series unit).

Assume, for example, that you have just bootstrapped a system that includes three disks: DP0, DP0F, and DZ0. The disk from which RDOS was bootstrapped automatically becomes the current and master directory.

For runtime convenience, RDOS offers the CLI's equivalence command, EQUIV, or system call .EQIV. Either version allows the user to change the global specifier of any tape drive or disk—except the master device—before initializing it. Thus, the developer can write programs using a generic, rather than a specific, name for a disk or tape device. At runtime, an available device is selected and its global specifier changed to the generic one via EQUIV. In the example that follows, the global specifier DP4 takes on the generic name DISK:

```
R
EQUIV DISK DP4 ⟨CR⟩
```

Now DP4 can be initialized under its new name and the program can be executed. When the device is released, RDOS restores its old specifier.

Referencing Disk Files

Because a file may exist in one of many subdirectories and a subdirectory may reside in one of many partitions, your CLI command or system call must indicate where RDOS can find this file. When more than one disk unit is present, you may need to enter a global specifier (eg, DP4) when initializing the directory that holds the file. A directory need only be initialized once with the INIT or DIR commands. Afterwards, the directory's name followed by a colon and filename will suffice, as shown earlier in Table 2.4. Assume, for example, that you want to execute file MYPROG.SV, in subdirectory SUBDIR, on secondary partition SECONDPART. Further assume that SUBDIR has not been initialized. (Otherwise, the statement SUBDIR:MYPROG ⟨CR⟩ would suffice.) You initialize SUBDIR, or any other directory, by entering the hierarchy of names in descending order, separating each from the next with a colon, for example:

```
R
INIT SECONDPART:SUBDIR ⟨CR⟩
```

Or, using the same format with the DIR command instead of INIT, you can designate the directory you want as the current directory, for example:

```
R
DIR SECONDPART:SUBDIR ⟨CR⟩
```

This statement initializes SUBDIR and makes it the current directory. All references to filenames that do not include directory specifiers are directed to the current directory. With one or more colons present, RDOS assumes that you want a file in another directory and searches for it there. The simple statement

```
R
MYPROG ⟨CR⟩
```

executes the program MYPROG.SV because SUBDIR is now the current directory.

Link Entries

The link entry allows a user in any directory to access any disk file by its name or by any other filename. Link entries are most often employed to save disk file space by allowing users in different directories to access a single copy of a commonly-used disk file. A link entry may point to other link entries, with a depth of resolution of up to ten. The file that is finally linked to is called the *resolution file*. Link entries are created with the CLI's LINK command or .LINK system call.

Creating a link entry is simple, requiring only its name be unique within its directory; the resolution file need not even exist when you do it. The link entry can have the same name as the resolution file, or not; it can exist on the same partition as the resolution file, or not.

The LINK command has two arguments:

```
LINK link-entry-name resolution-file-name
```

RDOS creates the link entry in the current directory unless instructed otherwise. It assumes that the resolution file resides in the current directory's parent partition—which can be either a secondary or the primary partition— not in a subdirectory. If the resolution file is elsewhere, its location must be indicated with one or more colons and specifiers.

Although the link entry need not have the same name as the resolution file, link operations are clearer and simpler if the entry shares a name with its resolution file. Link entries with different names are called *aliases*.

To use a link, the user or program must initialize the directory containing its resolution entry, along with all directories containing intermediate links. Moreover, the attributes of the resolution entry and of all intervening link entries must allow this operation. (The discussions of .CHATR and .CHLAT in Chapter 3 include the relevant attributes.)

As shown in Figure 2.5, two links exist to the resolution entry EDIT.SV on primary partition DP0. The resolution file, EDIT.SV, is the Text Editor utility supplied with RDOS systems. Normally, Data General utilities are loaded onto the master directory before system generation, and EDIT.SV is included among them. Because it is not in a subdirectory, linking to it is easy.

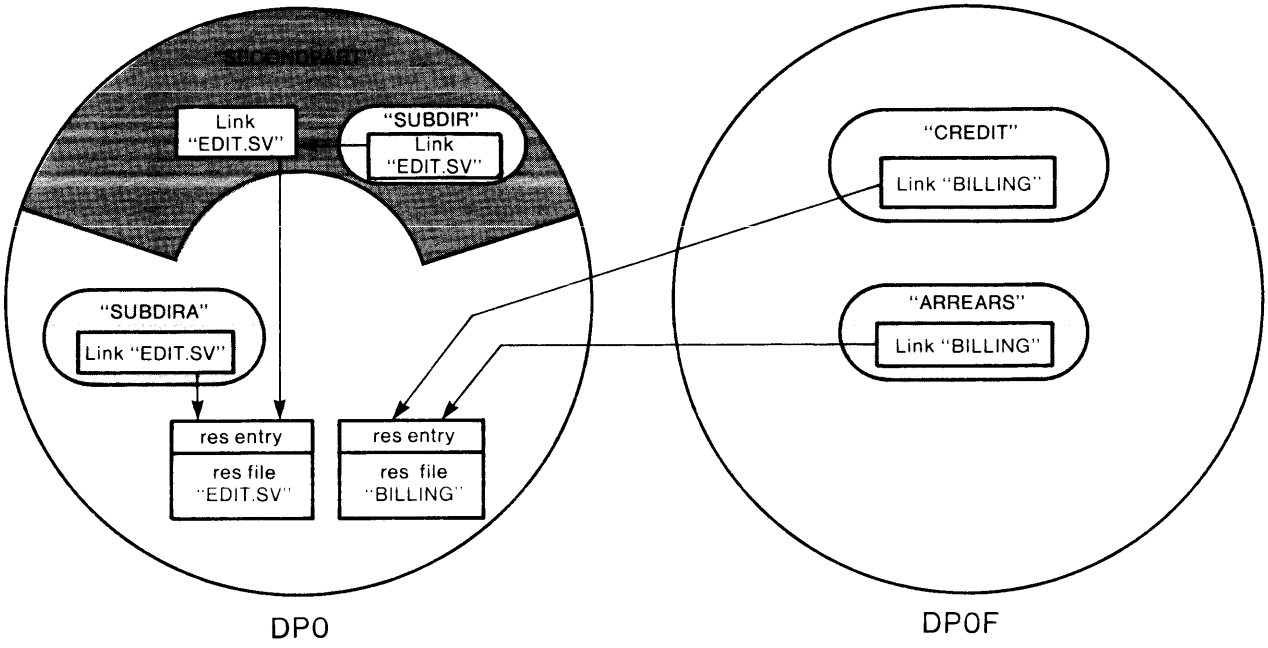


Figure 2.5 Link entries SD-00502

To recreate the structure of DP0 in Figure 2.5, you would enter the CLI command line

R
DIR SECONDPART ⟨CR⟩

to initialize the secondary partition and make it the current directory. The next statement creates a link to the resolution file, EDIT.SV

R
LINK EDIT.SV EDIT.SV ⟨CR⟩ or LINK EDIT.SV/2 ⟨CR⟩

where the first argument is the link entry's name, and the second, the name of the resolution file. The alternative command format containing the /2 switch directs RDOS to repeat the arguments twice. The result is an entry named EDIT.SV in the secondary partition that links to the Text Editor on DP0. This only works if you are in the parent directory. Otherwise you must specify a directory. For example, in the command line

LINK EDIT.SV UTIL: EDIT.SV

UTIL is the directory specifier. Users in partition SECONDPART can work with the editor while it occupies disk space on DP0 only.

The following command sequence creates a link from subdirectory SUBDIR to the Text Editor:

R
DIR SUBDIR ⟨CR⟩

R
LINK EDIT.SV DP0:EDIT.SV ⟨CR⟩ or LINK EDIT.SV/2 ⟨CR⟩

In a third command sequence, SUBDIRA is initialized, made current, and linked to EDIT.SV:

R
DIR DP0:SUBDIRA ⟨CR⟩

R
LINK EDIT.SV/2 ⟨CR⟩

The next series of commands creates two links—one in subdirectory CREDIT and the other in subdirectory ARREARS—from DP0F to file BILLING on DP0:

R
DIR DP0F:CREDIT ⟨CR⟩

R
LINK BILLING DP0:BILLING ⟨CR⟩

R
DIR ARREARS ⟨CR⟩

R
LINK BILLING DP0:BILLING ⟨CR⟩

Again, colons and specifiers are required if the resolution file does not reside on the partition that holds the current directory.

Before a link can be used, all intermediate links must be resolvable. This is accomplished by initializing all intervening directories. Figure 2.5 provides examples: if DP0 had not been initialized, neither link in DP0F would work; and if the link entry in SECONDPART were removed, the link in SUBDIR would be useless while the link in SUBDIRA would still function. Note that the CLI's UNLINK command or the system call .ULNK are the only ways to remove a link entry. The DELETE command and .DELET system call cause the link to persist and the resolution file to be deleted.

Each link entry is a filename whose sole function is to point to the resolution entry, or to a closer, intermediate link. Like other files, each resolution entry has a user file definition which includes two sets of attributes: (1) file access attributes, called *resolution entry attributes*; and (2) *link access attributes*.

Resolution entry attributes govern direct access to the file. They can be changed via the CLI command CHATR or the system call .CHATR, as explained in Chapter 3. The attribute N allows a link to exist but prevents anyone from using it. Other attributes govern reading, writing, renaming, or deletion. The A attribute makes permanent all other attributes of a resolution entry or file.

Link access attributes permit or restrict access to the resolution entry. Again, the N attribute forbids linking. The CLI command CHLAT or system call .CHLAT can be used to change these attributes.

Thus, although links to a resolution file are easily established, two sets of resolution entry attributes guard the resolution file. As seen by a link entry, the resolution file has a composite of link attributes and resolution entry attributes. More than one link entry may point to a resolution entry. Single user read-write opens and multiple read-only opens are allowed. In any command or system call, links and resolution filenames have the same effect. For an example, return to Figure 2.5 and assume that the current directory is CREDIT on DP0F. The statement

```
CRAND DP0:RATINGS ⟨CR⟩
```

creates a randomly organized file named RATINGS on DP0, as do the statements

```
R
LINK RATINGS DP0:RATINGS ⟨CR⟩

R
CRAND RATINGS ⟨CR⟩
```

After either set of commands, the current directory remains CREDIT and file RATINGS exists on DP0.

After creating and linking a file, all directories in the path to the resolution file must be initialized before that file can be opened with the .OPEN system call. Otherwise, the system returns error ERDNI (Directory Not Initialized) or error ERDSN (Device Not In System) from .OPEN, indicating that one or more intervening directories are uninitialized.

Note that the link entry offers much more than a simple way to share user files. A link entry can be created for any file, including a reserved device such as the line printer.

If a link is established to a file on magnetic tape, the device must be initialized before the link will work. A nondisk device cannot be linked, in turn, to another resolution file.

File Access Example

When a new disk is introduced to the system, only its primary partition exists. This section shows, by example, how a new disk might be organized according to the structures—partitions, subdirectories, and links—discussed earlier in this chapter. The example assumes that five users—two developers, two documentation specialists, and one support person—need space on one disk for their files. Ideally, each user would have as much disk space as needed; file space would be used efficiently; and each user's files would be safe from unauthorized access or alteration.

There are at least two obvious ways to approach these goals:

1. Create five secondary partitions and assign one user to each.

2. Create a single, large secondary partition and assign each person to a distinct subdirectory within it.

Both approaches protect all disk files while allowing each person to access files, such as utility programs, in the primary partition. The first option guarantees a fixed amount of file space to each user. A person who exhausts his or her space, cannot appropriate unused space on another person's partition. The second option allows each person to use as much file space as required from within the common secondary partition, as long as any unused space remains. Although this option guarantees no user a minimum amount of file space at any moment, it organizes file space more efficiently than the first alternative.

The best solution for this hypothetical installation adopts a middle ground, and is illustrated in Figure 2.6. The disk, DP0, is organized into one secondary position (DEVELOP.DR) for two application programmers, and another secondary partition (DOCUUMENT.DR) for two writers;

the fifth, more modest user works in a subdirectory (MARGE.DR) with files. Commonly-used public files are divided into two categories—system-related software, and all other utilities—and assigned discrete subdirectories (SYSTEM.DR and UTILITIES.DR). Users can link to these files from their directories, allowing an application program to run in the primary partition. A sample dialogue with this system's CLI follows Figure 2.6 to show how this organization might work in practice.

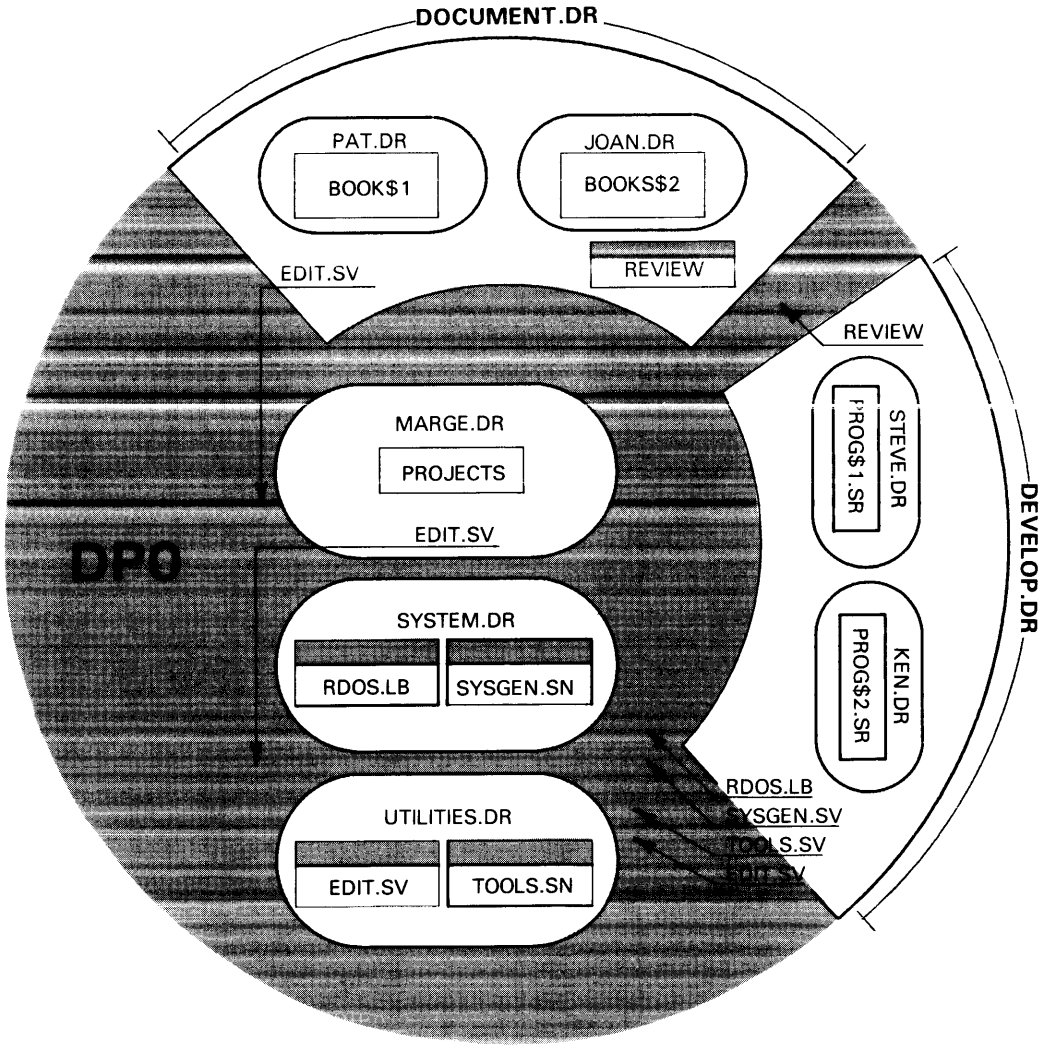


Figure 2.6 Sample organization of an RDOS disk

ID-00485

The first objective of this session is to obtain a line printer copy of file PROJECTS in subdirectory MARGE. After a bootstrap and log-on sequence, the CLI announces itself with the R prompt and the master directory, DP0, automatically becomes the current directory. The CLI's PRINT command is used to obtain line printer copies of a file:

```
R
PRINT MARGE:PROJECTS ⟨CR⟩
NO SUCH DIRECTORY :MARGE:PROJECTS
```

The CLI returned an error because, to RDOS, an uninitialized directory does not exist. The INIT command opens directory MARGE so that file PROJECTS can be printed.

```
R
INIT MARGE ⟨CR⟩

R
PRINT MARGE:PROJECTS ⟨CR⟩
```

Two directories, DP0 and MARGE are initialized at this time. The CLI's GDIR command shows which one is the current directory:

```
R
GDIR ⟨CR⟩
DP0
```

The next objective—to print a copy of BOOK$1—requires that secondary partition DOCUMENT and subdirectory PAT be opened.

```
R
DIR DOCUMENT:PAT ⟨CR⟩
NO MORE DCBS :PAT
```

The CLI's error message indicated that this RDOS system was generated to allow only three partitions to be initialized at any given time. Since DP0 and MARGE are already open, the addition of directories DOCUMENT and PAT brought the total to four.

The CLI's RELEASE command solves this problem by closing MARGE, allowing the DIR command to open two more directories and make PAT the current one. Once directory PAT has been opened, the CLI can print BOOK$1.

```
R
RELEASE MARGE ⟨CR⟩

R
DIR DOCUMENT:PAT ⟨CR⟩

R
PRINT BOOKS$1 ⟨CR⟩
```

The next sequence of commands creates a link from secondary partition DOCUMENT to the Text Editor program, EDIT.SV, contained in subdirectory UTILITIES. First the DIR command makes DOCUMENT the current directory. Then the LINK command creates a link entry named EDIT.SV that resolves to file EDIT.SV in the UTILITIES directory. Finally, the RELEASE command closes the current directory, causing DP0 to become current in its place.

```
R
DIR DOCUMENT ⟨CR⟩

R
LINK EDIT.SV UTILITIES:EDIT.SV ⟨CR⟩

R
RELEASE DOCUMENT ⟨CR⟩
```

As a result of this sequence, the Text Editor can be referenced and used from partition DOCUMENT, while the actual program occupies significant amounts of space in subdirectory UTILITIES only.

The objective of the last sequence is to back up all files in partition DEVELOP onto magnetic tape. First the DIR command makes this partition the current directory. Then the INIT command introduces the magnetic tape device to the system. Next, the DUMP command instructs RDOS to copy the contents of DEVELOP to the first file on magnetic tape. Note the use of the /V switch, which verifies each file on the console as it is copied.

```
R
DIR DEVELOP ⟨CR⟩

R
INIT MT0 ⟨CR⟩

R
DUMP/V MT0:0 DEVELOP ⟨CR⟩
 PROG$1.DR
*STEVE.DR
 PROG$2.DR
*KEN.DR
 EDIT.SV
 RDOS.LB
 REVIEW
 SYSGEN.SV
 TOOLS.SV
```

To conclude this session, the RELEASE command closes current directory DEVELOP, causing DP0 to become current in its place. Then DP0 itself is released, enabling an orderly shut-down of the operating system.

```
R
RELEASE DEVELOP ⟨CR⟩

R
RELEASE DP0 ⟨CR⟩
MASTER DEVICE RELEASED
```

Directory Command Summary

Table 2.3 summarizes the CLI commands and system calls used to manage disk files and directories. Chapter 3 discusses each system call in detail. The manual *RDOS/DOS Command Line Interpreter* provides more information on the CLI commands.

CLI Command	System Call	Meaning
CCONT	.CCONT	Create a contiguous file with all words zeroed.
CDIR	.CDIR	Create subdirectory.
CHATR	.CHATR	Change file attributes.
CHLAT	.CHLA	Change link access entry attributes.
CLEAR		Set a file's use count to zero.
	.CONN	Create a contiguous file without zeroing words.
CPART	.CPART	Create a secondary partition.
CRAND	.CRAND	Create a random file.
CREATE	.CREAT	Create a sequential file.
DELETE	.DELET	Delete a file.
DIR	.DIR	Specify a new current directory and initialize it if necessary.
EQUIV	.EQIV	Assign a new name to a global directory specifier, removing the old name or system name.
INIT	.INIT	Initialize and open a directory or device.
LINK	.LINK	Create link entry to a file in any directory.
RELEASE	.RLSE	Remove a directory or device from the system.
RENAME	.RENAM	Rename a file.
UNLINK	.ULNK	Delete a link entry.

Table 2.3 Directory command summary

Magnetic Tape Files

Data on magnetic tape can be accessed by both file and free form I/O. RDOS permits file access on nine- and seven-track magnetic tape, and supports up to 16 magnetic tape drives. The tape controller supports reading and writing at any density.

The operating system generates the following I/O modes:

Tape File I/O: 7-track 800BPI, even parity
9-track NRZI800BPI, odd parity
9-track PE 1600BPI, odd parity

Free Form I/O: Parity in any hardware combination except WRITE EOF is always even for 7-track, and odd for 9-track

If a controller detects an error during reading, the system makes ten attempts to reread the data before issuing error ERFIL, "File Data Error." If a data error is detected and returned to the CLI, the system displays the message PARITY ERROR: FIE MT*n*:*dd*, where *n* is the unit number and *dd* the file number.

When an error after writing is detected, RDOS attempts to backspace, erase, and rewrite up to ten times. The user receives an error message if the rewrite fails the tenth time.

An undefined error causes RDOS to return the tape status word as the error code. When this code is returned to the CLI, the message UNKNOWN ERROR CODE *n* is displayed, where *n* is the tape status word.

Nine and Seven Track Data Words

Under file and free format I/O, each data word output to nine-track units is written as two successive eight-bit bytes. Figure 2.7 shows how data is encoded on nine-track units.

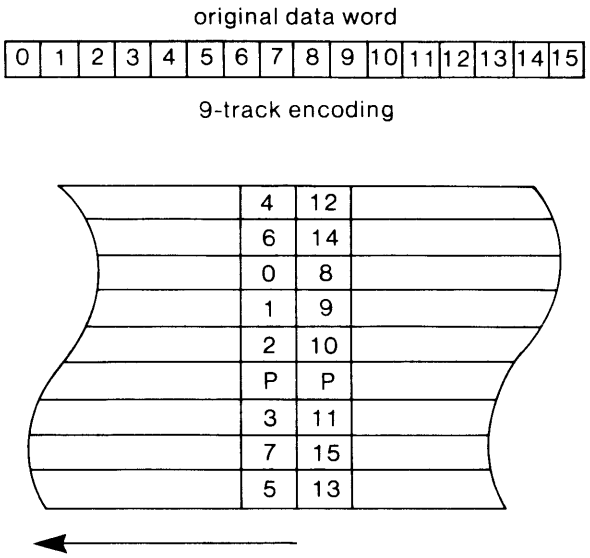
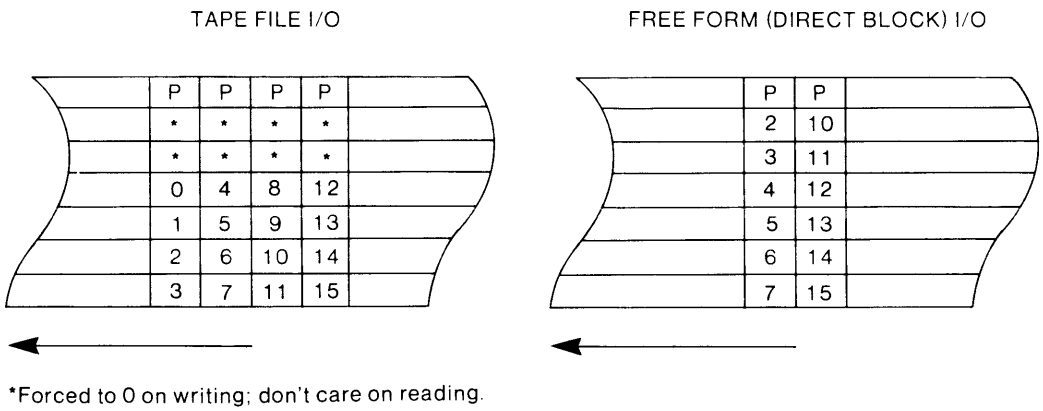


Figure 2.7 Data encoding (nine-track units) SD-00538

Data output to seven-track units is necessarily encoded in tape file I/O. RDOS encodes each 16-bit word as two data words, in four successive frames. The system encodes each word as two successive frames in free form I/O. Figure 2.8 shows how data is encoded on seven-track units.

Each tape has a physical end-of-tape (EOT) marker. An attempt to write beyond this marker causes RDOS to return the error ERSPC after completing the operation. A new file cannot be started beyond the physical end-of-tape marker. If you are writing to tape via the CLI's DUMP command and the system reaches the EOT mark, it stops writing and aborts the command. When writing on a system level, make sure that the reel holds enough tape to accept the file. If it reaches the physical end of tape (EOT) while writing, it will terminate writing to the tape.



*Forced to 0 on writing; don't care on reading.

Figure 2.8 Data encoding (seven-track units) SD-00539

Tape File I/O

In tape file format, RDOS writes and reads data in fixed-length blocks of 257, 16-bit words. It fills short blocks with nulls. Data files are variable in length; each one contains as many fixed-length blocks as the user needs. The first 255 words of each block contain user data, while each of the last two words contains the file number. Figure 2.9 shows how a data block is structured.

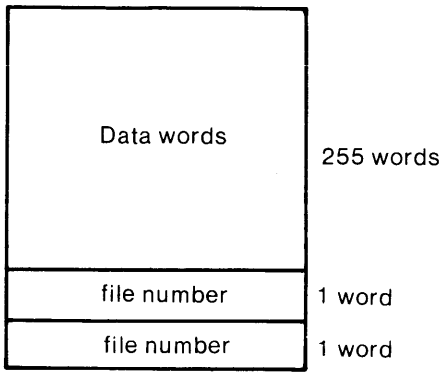


Figure 2.9 Data block structure SD-01032

RDOS writes a double end-of-file (EOF) mark after the first file on tape. The system begins writing at the first double EOF that it finds, overwriting the second EOF in the pair. After writing the file, RDOS leaves another double EOF at the end of it. The system writes files in consecutive order, starting with file number 0 and continuing through file number 99.

Free Form I/O

In addition to tape file I/O, RDOS allows users to read and write data to magnetic tape in free format, record by record. The system call .MTOPD opens a tape unit for free form. Data is read or written via the system call .MTDIO. (Both calls are fully described under "Input/Output Commands" in Chapter 3.)

Essentially, .MTDIO allows a program to read or write from two to 4096 words within a data record, and to space forward or backward through one to 4096 data records or to the start of a new data file. Additionally, this call allows the program to rewind a reel, write an end-of-file mark, read the transport status, and perform other machine-level operations. Under free form I/O, the system does not maintain a tape file pointer after it locates the file specified in .MTOPD.

Initializing and Releasing a Tape Drive

The CLI's INIT command initializes a tape drive and rewinds the tape on that unit, for example, INIT MT0 ⟨CR⟩. Full initialization with the INIT/F command rewinds the tape and writes two EOFs (logical end-of-tape indications) at its beginning. The INIT/F command must be executed on all new magnetic tapes before they are used. Note that this command effectively erases a tape by permitting the system to overwrite all files on it.

The CLI's RELEASE command rewinds a tape and releases its drive from the system.

Referencing Tape Files with File I/O

Files are placed on tape in numeric order, beginning with file number 0. A tape that is long enough can contain up to 100 files, the last having file number 99.

To access a tape file in a CLI command line, enter the command followed by the tape specifier and drive number, colon, and a file number. In this statement, for example,

```
R
PRINT MT0:6 ⟨CR⟩
```

MT is the specifier for magnetic tape, 0 is the unit number, and 6 is the file number. All tape specifiers have the format MT*n*:*m* or CT*n*:*m*, where *n* is a drive number between 0 and 17 octal and has no leading zero, and *m* is a file number in the range of 0 through 99. No leading zero is needed to enter the first 10 file numbers. Thus, file number 8 on magnetic tape unit 2, could be represented as MT2:08 or MT2:8. Both the global tape specifier and file number must be entered. Otherwise the system responds with an error message, ILLEGAL FILE NAME.

The following examples reference files on magnetic tape and disk from the CLI:

```
R
DUMP MT0:0 ⟨CR⟩
```

Dump all nonpermanent files from the current directory onto tape. (This statement is commonly used to perform magnetic tape backups.) The disk files will comprise file number 0 of the tape on unit 0 when this command line is executed.

```
R
LOAD MT0:0 ⟨CR⟩
```

Load the files from tape file 0 into the current disk directory. Note that the LOAD command transfers only files that have been previously dumped with the CLI's DUMP command. Likewise, the FLOAD command transfers only files that have been previously dumped with the CLI's FDUMP command. The XFER command must be used to transfer any file that is not in DUMP format.

```
XFER MT0:0 DATABASE ⟨CR⟩
```

Transfer the contents of the first file on tape unit 0 to DATABASE, the current disk directory.

Files must be written on magnetic tape in numeric order. Assume, for example, that you have transfered a disk file to tape unit 0, which contains a new, fully initialized tape. The command line

```
R
XFER SOURCEFILE MT0:0 ⟨CR⟩
```

posits SOURCEFILE on tape as shown in Figure 2.10. The system recognizes only file numbers 0 and 1 on this tape; that is, because RDOS assigns numbers incrementally, only these numbers exist. An attempt to reference any other file on the tape would result in an error message, FILE DOES NOT EXIST:MT0:*n*, because file 0 is the last file on this tape.

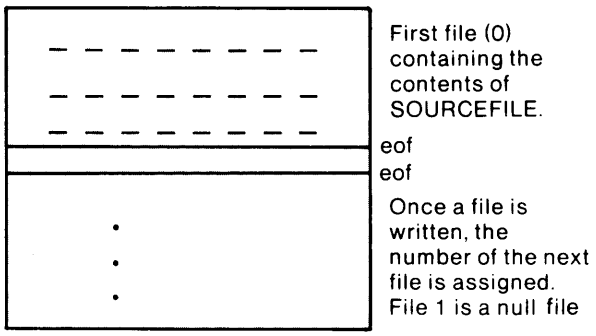
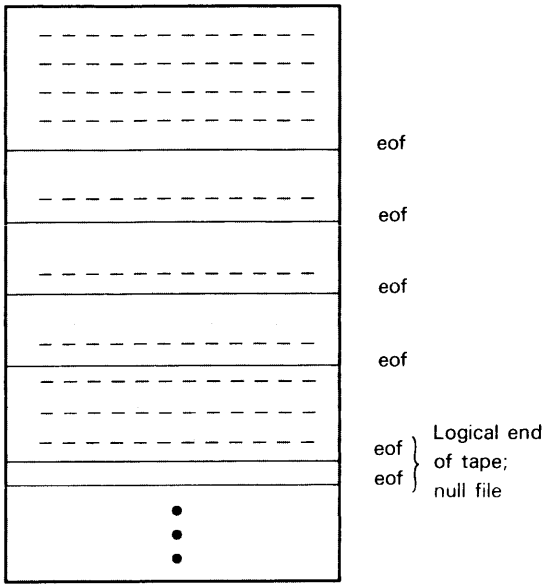


Figure 2.10 Writing the first tape file DG-25453

Users are advised to make a note of each file number when writing files on tape. Otherwise, a file may be inadvertently overwritten and destroyed, along with all subsequent files. Assume, for example, that a tape on drive 0 contains four files, as shown in Figure 2.11. The XFER command overwrites the contents of file 1 with MYFILE, voiding the location data of files 2 and 3 in the process. As a result, the original file 1 and all subsequent files are lost.

1 A tape file on drive 0 contains four files.



2 User issues XFER MYFILE MT0:1 %pi(22)

3 Command overwrites contents of file 1 with MYFILE. The original file 1 and all subsequent files are lost.

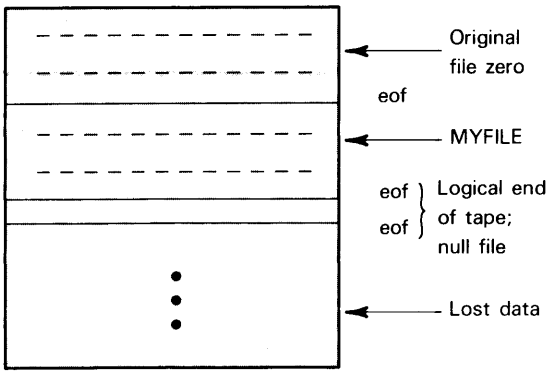


Figure 2.11 Overwriting tape files ID-00487

Before physically removing a reel of tape, its transport must be released with the CLI's RELEASE command. This command rewinds a tape and resets the system's tape file pointer to file 0 for correct file access in the future. The implications of the logical end-of-tape mark employed by RDOS should also be noted. For example, a user who deliberately writes a null file can write no other files to that tape; the null file becomes the last file.

Linking to Tape Files

Links can be established from disk files to resolution files on tape using the linking mechanism described earlier. The act of linking disk file STATISTICS to tape file MT0:0 creates a link entry in the current disk directory for resolution file MT0:0; the link entry to file MT0:0 is named STATISTICS. References to this file in the current disk directory are resolved as references to file MT0:0.

Multiplexors

The SYSGEN program allows users to specify multiplexors and their characteristics. RDOS supports three kinds of Data General multiplexors:

1. Type 4255-4258 Asynchronous Line Multiplexor (ALM), with device code 34_8 for the primary unit and 44_8 for the secondary unit. The ALM supports from one to 64 full- or half-duplex lines.

2. Type 4060-4063 Asynchronous Communications Multiplexor (QTY), with device code 30_8 for the primary unit and 70_8 for the secondary unit. The QTY supports from one to 64 full- or half-duplex lines.

3. The Universal Line Multiplexor (ULM), with device code 34_8 for the primary unit and 44_8 for the secondary unit. The ULM handles up to one synchronous and four asynchronous, full-duplex lines.*

A full-duplex line allows data to flow two ways simultaneously: users can transmit to RDOS over it, while RDOS transmits to users' terminals. Although RDOS assumes full-duplex lines, half-duplex protocols can be incorporated if desired.

Each ALM, ULM, or QTY line is a filename, of the form QTY:*x* where *x* is a number from 0 to 63. A multiplexed line can be opened on any RDOS I/O channel; the system calls .RDL/.WRL and .RDS/.WRS are used to read and write to it. (Chapter 3 explains how to select a channel number, open files, and use the read/write calls.) No more than one read and one write may be outstanding on any single line. The system call .CLOSE must be used to close a line and abort I/O, since the task call .ABORT does not affect QTY/ALM I/O.

When you open a multiplexed line, or any other file, the contents of AC1 determine what operations RDOS will allow on it. AC1 acts as a characteristic disable mask, as described in Chapter 3. Table 2.4 lists the characteristic bits that affect multiplexors.

*RDOS does not support the synchronous lines; rather, other software available with RDOS, such as the Communications Access Manager, supports them.

AC1	Meaning
DCEDT = 1B0	Masking allows editing features such as rub-out and backslash to work even when echoing is suppressed.
DCCRE = 1B4	Masking disables carriage return echoes on line reads. (CR then acts as enter key.)
DCLAC = 1B6	Masking disables line feed after CR.
DCPCK = 1B7	Masking disables software parity on QTY; no effect on ALM or ULM.
DCXON = 1B8	Masking enables XON/XOFF protocol for $TTR. (This protocol prevents the teletypewriter reader from overflowing the multiplexor read buffer.)
DCNAF = 1B9	Masking disables 20 nulls after line feed.
DCKEY = 1B10	Masking disables echo, CTRL-Z (end-of-file), CTRL S, and CTRL Q, along with line and character rubout.
DCTO = 1B11	Masking enables backspacing for rubout on CRT displays only. For QTY/ALM/ULM lines, the Newline key is treated as a carriage return.
DCLOC = 1B13	Masking makes this a modem line.
DCCGN = 1B14	Masking disables TAB expansion.
DCNI = 1B15	Masking enables multiplexor interrupts.

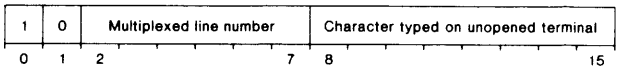
Table 2.4 Characteristic bits that affect multiplexors

When AC1 equals zero on the .OPEN, the multiplexed console has the following default characteristics:

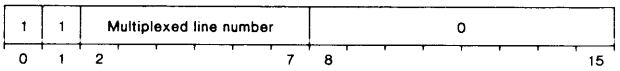
- Line feeds after carriage returns.
- Twenty nulls after line feed.
- Characters echoed during line reads. SHIFT-L (\ ) deletes line. RUBOUT deletes character and is echoed as ←. CTRL-Z and ESCAPE also result in end-of-file error.
- This is a local line.
- TABS are expanded as spaces.

Line 64 Reads

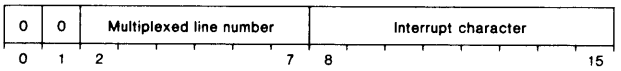
RDOS allows you to monitor activity on all unopened multiplexed lines, and to monitor console interrupts from all opened multiplexed lines. If a task opens QTY:64 and issues a read line or read sequential call, RDOS suspends this task until (1) a user presses a key at the end of an unopened line, or (2) a user hits an interrupt on an opened line. After receiving the character typed, RDOS readies the task, takes the normal return from the read call, and passes the following data in AC2:



When RDOS receives and answers a ring from a modem, it sends the following data to line 64, in AC2:

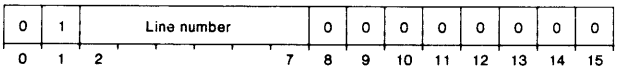


This data allows your program to detect a service request from a distant terminal. If the request comes from an unopened line, your program can then .OPEN the line for communications. QTY:64 can be opened by a foreground and a background task; when this occurs, each task receives characters from unopened lines. If an open line receives an interrupt (CTRL-A and CTRL-C are defaults), RDOS readies the task that opened line 64 and passes the following data in AC2:



The system generation program allows you to select interrupts other than CTRL-A and CTRL-C. A task receives and interrupts from an opened line only if it is in the ground that opened that line's channel.

If an open line receives a hangup notification, RDOS returns the following data in AC2:



Line 64 Writes (ALM and ULM only)

RDOS allows you to change the device characteristic disable mask, line speed, or modem state on any ALM line. To effect these changes, issue the system call .WRL to a channel opened on QTY:64 and pass the following data.

To change the mask (on opened lines only):

AC0 = W64DC + line number
AC1 = new mask.

To change the line speed:

AC0 = W64LS + line speed
AC1 = new line speed (0, 1, 2, or 3 for ALM clock; 1 through 15 for ULM line code, as shown in Table 2.5.)

To change the modem state:

AC0 = W64MS + line number
AC1 = [W64DTR][+][W64RTS]

These bracketed entries are optional. W64DTR raises Data Terminal Ready; if you omit it, DTR is lowered. W64RTS raises Request To Send; without it, RTS is lowered.

To change any or all characteristics on any line:

AC0 = W64CH + line number
AC1 = new characteristic mask

These symbols are defined in the user parameter file, PARU.SR, which is listed in Appendix B. Note that RDOS does not check the validity of user input, requiring that you exercise care when changing the characteristics of an open line.

ULM Line Codes

During system generation, a line speed is selected for all ULM lines. Subsequently, a user can change the line speed of any ULM line via the line 64 write mechanism described earlier. Table 2.5 lists the 15 (decimal) ULM codes, one of which you will specify in AC1 to select the matching line speed.

ULM code	Matching line speed
1	19200
2	50
3	75
4	134.5
5	200
6	600
7	2400
8	9600
9	4800
10	1800
11	1200
12	2400
13	300
14	150
15	110

Table 2.5 Selecting a ULM line speed

Multiple Channels

A ground may have several channels opened to the same line. Except for line 64, however, the same line cannot be opened in both grounds. The first channel opened on a line becomes the master channel, and all other channels opened on this line become subordinate to it. Closing the master channel prevents subordinate channel numbers from using the line. Each subordinate channel must be closed before it can be opened again, or reassigned, on another line. If you open a new channel on a line after closing the master, the new channel becomes the master channel.

Modem Support Under RDOS

A modem control interface allows software to be written that controls various asynchronous modems. These modems must support a subset of the EIA RS-232C interface standard. A modem must supply the following signals:

- Receive Data
- Clear to Send (may be strapped to DSR or RTS)
- Ring Indicator
- Carrier Detect *or* Data Set Ready

Any inactive signal that is wired in the interface cable should be properly terminated to avoid false activation.

The modem must be fully operative by controlling only the Transmit Data, Request to Send, and Data Terminal Ready signals. In situations where remote consoles are connected to a DG system via the Bell switched voice network, the modem must also have full-duplex and auto-answer capabilities; transmit and receive data at equal rates; and drop Data Terminal Ready low, forcing a disconnect.

If standard modem timer software is selected during system generation, the modem must supply the Carrier Detect signal, which ensures proper handling of connect and disconnect procedures by RDOS. If the standard modem timer is not included, the modem must provide the Data Set Ready signal. Further, it should be capable of raising this signal after Carrier Detect and lowering it after a disconnect.

It is essential that the modem be able to recognize a disconnect and drop Data Set Ready. Loss of Carrier Detect is not sufficient to determine when to drop this signal. Bell modems use direct current on the phone line as an indication of a disconnect, and drop Data Set Ready as a result. If the modem does not provide this function, the line will appear busy to the next caller if the previous caller has hung up.

When RDOS is bootstrapped, it raises DTR and RTS unless you have changed the ALM parameter file (ALMSPD.SR) to drop either or both signals. On a ring interrupt, RDOS raises both DTR and RTS; on a disconnect, if DSR is low, it lowers DTR and CTS.

When a modem's Data Set Ready signal is low, it cannot communicate. In this case, RDOS takes the error return on all reads and writes to its modem line, and places code ERRDY in AC2. Note, however, that the error return occurs only if you defined the line as a modem line by masking DCLOC on the .OPEN.

Multiplexor Error Messages

Table 2.6 lists the errors that relate to reads and writes on multiplexed lines. Other read/write errors are described in Chapter 3.

AC2	Mnemonic	Meaning
24	ERPAR	Parity error detected on read.
47	ERSIM	Duplicate read or duplicate write.
127	ERRDY	Line not ready; modem's DSR is low.
130	ERINT	Console interrupt received.
131	EROVR	Hardware overrun error on read.*
132	ERFRM	Hardware framing error on read.*

Table 2.6 Multiplexor error messages
*This error clears the read buffer and errors the read request.

ALMSPD.SR

The source file ALMSPD.SR defines the characteristics of each line of the ALM or ULM. This source file can be edited with the Text Editor and assembled with MAC, the macroassembler utility, to tailor multiplexed lines for specific applications. The new line specifications are contained in the binary file ALMSPD.RB and incorporated by the system generator when it builds a new RDOS system. If you do not define a line in this module, or if you set its characteristics at default, the line has these characteristics:

clock frequency (ALM) or line speed (ULM) as defined to SYSGEN

one stop bit

seven bits per character

even parity

no loopback

signals DTR and RTS raised on initialization

no modem support

no carrier monitoring protection

These characteristics can be defined for any line by entering the statement

```
LNDEF xx,DEFAULT
```

in ALMSPD.SR, where xx is the two-digit, decimal number for the line to be set. To define unique line characteristics, insert a line of the form

```
LNDEF xx,spd,stop,bits,par,loop or
```

```
LNDEF xx,spd,stop,bits,par,loop,dtr,rts where
```

xx is the two-digit decimal line number

spd is the clock frequency (may be 0, 1, 2, or 3 for ALM clock or 1 through 15 for ULM line speed

stop is the number of stop bits per character (may be 1 or 2)

bits is the number of bits per character (may be 5, 6, 7, or 8, not including the parity bit)

par indicates whether you wish no parity to be generated or checked (specify NO), even parity (EVEN), or odd parity (ODD)

loop indicates whether you want to enable loopback (specify LOOPBACK or NOLOOPBACK)

dtr defines the state of Data Terminal Ready on initialization (DTRHIGH or DTRLOW)

rts defines the state of Request To Send on (RTSHIGH or RTSLOW)

Note that the arguments for dts and rts may be omitted if you wish to set their states high. The following example defines the characteristics of ALM line 3, including a clock frequency of one; two stop bits; seven bits per character; even parity; and no loopback. Upon initialization, the DTR and RTS signals will be high.

```
LNDEF 03,1,2,7,EVEN,NOLOOPBACK
```

The next example determines that ULM line 4 will run at 4800 baud, and have one stop bit, seven bits per character, odd parity, and no loopback. Again, the DTR and RTS signals will be initialized high.

```
LNDEF 04,9,1,7,ODD,NOLOOPBACK
```

After defining ALMSPD.SR, execute the command line MAC ALMSPD $LPT/L ⟨CR⟩ before generating a new RDOS system.

Chapter 3

Single-task Programming

This chapter describes most of the system calls needed to program under RDOS in a single-task environment. It explains system and task command structures, summarizes the most commonly-used system calls, and then discusses individual, single-task calls under the following headings:

- Device and Directory Commands
- File Maintenance Commands
- File Attribute Commands
- Link Commands
- Input/Output Commands
- Console I/O Commands
- Memory Allocation Commands
- Device Access Commands
- Clock and Calendar Commands
- Spooling Commands
- Keyboard Interrupt Commands

In conclusion, the system calls described in this chapter are summarized in table form.

Readers will find further information on single-task programming in Chapter 4, where program swaps and overlays are discussed; in Chapter 5, which covers system clock commands used in single-task environments; in Chapter 6, which explains how to run in two grounds; and in Chapter 7, where user interrupts are explained.

Multiple and Single-task Environments

A program task is an execution path through user address space that uses system resources such as I/O, overlays, or simple CPU control. User address space includes all memory from location 16_8 through NMAX—1.

In a single-task environment, the program itself is the only task. A program initiates a multitask environment by creating a task via task calls .TASK or .QTSK. In planning a multitask program, you must specify multiple tasks with assembly language pseudo-ops or with RLDR switches. Then RLDR will copy the multitask scheduler, called TCBMON, into your program and allot the number of Task Control Blocks (TCBs) specified.

If you omit task and I/O channel pseudo-ops and task/channel switches, RLDR assumes a single-task program and copies the single-task scheduler into it. RLDR also allots eight channels — enough for most single-task programs. Either a single- or multitask program can use all system calls described in this chapter. For more information on multitasking, consult Chapter 5.

Note that the task scheduler and other modules differ for certain kinds of systems (eg, unmapped and mapped NOVA and ECLIPSE systems), meaning that programs loaded under one type of system may not execute on a system of another type. When loading for a different system, obtain the appropriate system library (SYS.LB) for the target system and ensure that RLDR searches it, rather than the current library, during the loading process. This procedure is most easily accomplished from a subdirectory that contains the target system's library and links to RLDR.

System and Task Calls

RDOS system and task calls allow users to communicate directly with the operating system; they are similar, but not identical.

You begin each system call with the mnemonic .SYSTM, which assembles as a JSR @ 17 instruction. This instruction enables the system to respond to your command. After executing a system call, the system (1) takes a normal return to the second instruction after the command word, or (2) takes the error return to the first instruction following the command word, if an exceptional condition is detected. System calls always reserve AC2 for the error code.

Descriptions of system calls in this manual abide by the following, generic format:

ACn - Required input to the call.

```
.SYSTM
command
error return
normal return
```

On an error return, RDOS passes an error code in AC2. On a normal return, each accumulator, except AC3, is restored unless used to return output.

ACn - Output from the call.

AC3 - The content of location 16 (the User Stack Pointer) is the default value.

Required input for many system calls includes a byte pointer to a specific filename. When you include this byte pointer, the filename pointed to may include directory specifiers as well.

A task call resembles a system call, with these exceptions: (1) you enter no .SYSTM mnemonic before the task command word; (2) RDOS executes task calls in user address space, not in system space; and (3) task calls that cannot take an error return do not reserve an error return location. Almost all system calls reserve an error return location, even if no error return is possible. The comands in this chapter are all system calls.

Status On Return From System Calls

This discussion summarizes the status of the accumulators upon return from a system or task call.

For certain calls, the system returns information in AC0, AC1, and/or AC2; if it does not, the carry and all accumulators except AC3 are preserved. The system always returns the contents of location 16_8 (the USP) in AC3 by default, unless you specified a particular module, such as ESAC3, in the RLDR command line. Thus, if you loaded a program with module N3SAC3 (NOVA3s only), AC3 would contain the contents of the frame pointer register upon return from a call. Similarly, if you loaded a program with module ESAC3 (ECLIPSEs only), the system would return the contents of location 41_8, the frame pointer, in AC3. On error returns, RDOS uses AC2 to return numeric error codes, which are listed in Appendix A.

> **NOTE:** *In this book, the error codes associated with each system call represent the most common errors only; their meanings have been expanded and interpreted in light of the call.*

I/O Channel Numbers

Before a file can be accessed for I/O, it must receive an I/O channel number in your open call. The file retains its channel number while it is open, and must be accessed via this number instead of the filename. The associated channel number is released when you close the file. An I/O channel number immediately follows the call word in your program. Thus, if the channel number is n, the I/O calls for a file could run as follows:

```
open n
.
.
.
file reads/writes n
.
.
.
close n
```

In a mapped system, the number of foreground and background channels is specified during system generation; the maximum for each ground is 377_8. SYSGEN asks no question about channels for an unmapped system, whose maximum of 377_8 is predefined. RLDR allots eight I/O channels, numbered 0 through 7, for a single-task program. Although this number is generally sufficient, you may want to specify more channels using the RLDR program's /C switch or the macroassembler's pseudo-op, .COMM TASK.

Selecting a Channel

There are two ways to assign a channel number to a file: either directly when you open, for example,

```
.OPEN 3
```

or via AC2. If your opening specifies number 77 (or CPU), RDOS opens the file on the channel number contained in the right byte of AC2. To open on a number above 77— assuming that your program permits one— you must open on 77 and pass the number in AC2.

The major advantage to opening on 77 is that the system call .GCHN can be used to find a free channel for your open. .GCHN returns the number of a free channel in AC2. This number can be assigned a name that identifies the channel for all I/O to the file. Unless all channels are in use, this method ensures a free channel for file I/O. The following example demonstrates:

```
.SYSTM
.GCHN
JMP ER
STA 2, FILE1        ;STORE THIS CHANNEL
                    ;NUMBER UNDER "FILE1".
.
.
.
LDA 2,FILE1
.SYSTM
.OPEN 77            ;OPEN "FILE1".
JMP ER
.SYSTM
.WRS 77             ;WRITE TO "FILE1"
.
.
.SYSTM
.GCHN
JMP ER
STA 2, FILE2        ;STORE NUMBER UNDER "FILE2".
.SYSTM
.APPEND 77          ;OPEN "FILE2" FOR APPENDING.
JMP ER
.
.
.
```

Commonly Used Commands

In the process of developing application programs, you will use certain system calls quite frequently and others rarely or not at all. Table 3.1 attempts to summarize the most useful calls in the sequence that you might use them in a program. Each call is described by name, format, accumulator data, and possible error codes. The table assumes that you will use the CLI to create and initialize partitions and subdirectories, to execute magnetic tape I/O, and to control spooling. It further assumes that your program will not attempt to alter file attributes, create link entries, or manage a multitask environment, although these objectives can be accomplished via RDOS system calls if you choose.

Note that Table 3.1 assumes a single-task environment; it does not cover foreground/background calls (Chapter 6) or multitasking (Chapter 5). Each file I/O command requires a channel number, as indicated; the term *btptr* means byte pointer; and each system call has the generic form:

```
.SYSTM
call name
error return to program
```

The following example shows this format in practice:

```
                    LDA 0,BTPTR
                    .SYSTM
                    .DIR
                    JSR ERROR
                    .
                    .
                    .
BTPTR:              .+1*2
                    .TXT "DP1:SUBDIR"
ERROR:              .SYSTM
                    .ERTN
                    JMP.—2
```

Call	Purpose	Remarks
.CRAND	Creates a random file.	AC0: btptr to filename.
.CCOND	Create a contiguous file.	AC0: btptr to filename. AC1: number of disk blocks for the file.
.OPEN n	Open a file for I/O on channel n.	AC0: btptr to filename. AC1: characteristic disable mask. You can specify the system default mask (normal procedure) by passing 0 via a SUB 1,1 instruction before the .OPEN.
.APPEND n	Opens a file for appending, on channel n. Sets position for writing at the end of the file.	AC0: btptr to filename. AC1: characteristic disable mask. As with .OPEN you can specify the system default mask before the .APPEND.
.RDL n	Reads an ASCII line on channel n. Counterpart of .WRL command.	AC0: btptr to area large enough for line (133 maximum). AC1: returns the count of characters read.
.RDS	Reads sequentially from the file opened on channel n. Sequential mode is required for binary data.	AC0: btptr to starting byte address of data. AC1: number of bytes to be read.
.WRL n	Writes an ASCII line to the file opened on channel n. Writing begins at start of file if you opened it with .OPEN; at end if you opened it with .APPEND. Limit is 132 characters terminated by a CR, null, or form feed.	AC0: btptr to area that holds the ASCII line.
.WRS n	Writes sequential bytes to the file on channel n. See the .WRL command for position information.	AC0: btptr to starting byte address of data. AC1: number of bytes to be written.
.WRB n and .RDB n	Direct block I/O calls that write or read a series of disk blocks to or from the random or contiguous file on channel n.	AC0: starting address for the block write or read. AC1: starting relative block number in the series. AC2: left byte number of 256-word blocks to be written or read to the file.
.CLOSE n	Closes the file opened on channel n. RDOS then updates the file's UFD information. (.ERTN and .RTN close all channels in the current program.)	
.DELET	Delete a file.	AC0: btptr to filename.

Table 3.1 Commonly used commands

Table 3.2 lists the system calls that control NMAX, that execute and return from program swaps or chains, and that load overlays. Table 3.3 lists the error codes that AC2 may contain if your program takes the error return from any of these system calls.

Call	Purpose	Remarks
.MEM	Return the current program's NMAX value in AC0, along with the value of the highest memory address available for user programs in AC1.	
.MEMI	Raise NMAX to the value entered in AC0, or lower NMAX by the value entered in two's complement in AC0. RDOS returns the new value in AC1.	
.ERTN or .RTN	Close the channels in the current program and return to (resume execution of) the next higher-level program (usually the CLI). .ERTN returns an error code in AC2; if return is to the CLI, it also prints an error message on the console.	
.OVOPN n	Open overlay file for reading on channel n. Before your program can use overlays, you must open them on a channel. You close the channel via a .CLOSE n.	AC0: btptr to overlay filename, including .OL extension.
.OVLOD n	Load an overlay from the overlay file opened on channel n into its reserved memory node.	AC0: overlay descriptor. AC1: conditional load flag.

Table 3.2 Calls that control memory, returns and overlays

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number (legal range is 0 through 377_8.
1	ERFNM	Illegal filename (only alphanumeric or $ characters are permitted).
3	ERICD	Illegal command for device (for example, trying to read from the line printer).
6	EREOF	End of file detected while reading, or attempt to write beyond the end of a contiguous file.
7	ERRPR	File is read-protected.
10	ERWPR	File is write-protected.
11	ERCRE	File already exists.
12	ERDLE	File (directory) does not exist.
13	ERDE1	File cannot be deleted because it has the permanent attribute.
15	ERFOP	File has not been opened.
21	ERUFT	This channel is in use.
22	ERLLI	Line limit (132 characters) exceeded.
26	ERMEM	Attempt to allocate more memory than is available.
27	ERSPS	File space exhausted in current partition.
33	ERRD	Attempt to read or write into system space (unmapped systems only).

Table 3.3 Possible errors from calls that control memory, returns and overlays

AC2	Mnemonic	Meaning
36	ERDNM	Device not in system.
37	EROVN	Illegal overlay number.
40	EROVA	File not accessible by direct block I/O.
47	ERSIM	Simultaneous reads or writes attempted to same QTY/ALM line.
52	ERIDS	Illegal directory specifier.
66	ERDNI	Directory not initialized.
74	ERMPR	Address outside address space (mapped systems only).
101	ERDTO	Disk timeout occurred.
103	ERMCA	This MCA channel is in use.
104	ERSRR	A short receive request terminated the MCA transmission.
106	ERCLO	MCA/QTY/ALM output terminated by channel close.
124	ERZCB	Attempt to create a contiguous file of zero length.

Table 3.3 Possible errors from calls that control memory, returns and overlays (continued)

Device and Directory Commands

This section describes the RDOS system calls that pertain to opening and releasing disk and magnetic tape drives and disk directories; it also covers the commands that create disk partitions and subdirectories. In order of discussion, these commands are:

.INIT	Initialize a directory or device
.DIR	Select a different current directory
.RLSE	Release a directory or device
.GDIR	Get the current directory's name
.CDIR	Create a subdirectory
.CPART	Create a secondary partition
.EQIV	Temporarily rename a nonmaster device or tape drive
.GSYS	Get the current RDOS system's name
.MDIR	Get the master directory's name

.INIT

Initialize a directory or device

A program can initialize devices and directories via the system call .INIT. When this command is invoked and AC1 does not contain −1, a partial initialization of the device or directory occurs, making all files in the directory available to the system software, as a result. Partial initialization of a magnetic tape rewinds the tape and resets the tape file pointer to file zero. If AC1 contains 177777 when you invoke .INIT, a full initialization of the device results. Full initialization of a magnetic tape rewinds the tape and writes two EOFs to signify the logical end-of-tape. All files on that tape are lost as a result. Full initialization of a disk builds new system (SYS.DR) and map (MAP.DR) directories on it, effectively destroying all existing files. RDOS treats full initialization of a secondary partition or subdirectory as a partial initialization.

Required Input

AC0 - Byte pointer to a directory or device specifier.

In each byte pointer, bits 0—14 contain the word address that holds or will receive the byte. Bit 15 specifies which half (0 left, 1 right).

Format

```
.SYSTM
.INIT
error return
normal return
```

Possible Errors

AC2	Mnemonic	Meaning
1	ERFNM	Illegal filename
10	ERWPR	Device is write-protected (full initialization only).
12	ERDLE	Directory does not exist.
27	ERSPC	Out of disk space.
31	ERSEL	Unit improperly selected.
36	ERDNM	Device not in system.
45	ERIBS	Insufficient number of Device Control Blocks (DCBs) specified during system generation.
51	ERNMD	Same as above.
52	ERIDS	Illegal directory specifier.
56	ERDIU	In a dual processor system using an IPB: the other CPU is using this directory.
57	ERLDE	Link depth exceeded.
74	ERMPR	Address outside address space.
77	ERSDE	Error detected in SYS.DR of nonmaster device.
101	ERDTO	Disk timeout occurred.
102	ERENA	No linking allowed (N attribute).
112	EROVF	Too many chained directory specifiers caused system stack overflow. Occurs only when links are used in the specifier string.
121	ERFMT	Disk format error. Try to dump the disk and run DKINIT on it.
122	ERBAD	Disk has invalid bad block table. Dump the disk and run DKINIT on it.

.DIR

Initialize a directory or device

When you bootstrap an RDOS system, the directory that holds the system becomes the current directory. The .DIR command selects a different current directory, and—provided that directory has not been initialized—performs a partial initialization.

After invoking the .DIR command, you can access all files in the new directory without using directory specifiers. .DIR is not mandatory for file access in nonmaster directories, however, since RDOS permits directory specifiers in all filename arguments to system commands. The following example shows how a directory specifier isused to access MYFILE, in DP4, from master directory DP0F.

```
                .TXTM 1
                  .
                  .
                  .
                LDA 0,  .MYFILE
                  .
                  .
                  .
MYFILE:         .+1*2
                .TXT    "DP4:MYFILE"
```

The next example invokes the .DIR command to achieve the same results.

```
                .TXTM 1
                  .
                  .
                  .
                LDA 0,  .DP4
                .SYSTM
                .DIR
                  .
                  .
                  .
                LDA 0,  .MYFILE
                  .
                  .
                  .
.DP4:           .+1*2
                .TXT  "DP4"
.MYFILE:        .+1*2
                .TXT    "MYFILE"
```

In the first example, DP0F remained the current directory; in the second, the .DIR command made DP4 the current directory.

Required Input

AC0 - Byte pointer to directory name string.

Format

```
.SYSTM
.DIR
error return
normal return
```

If RDOS takes the error return, the current directory definition remains unchanged.

Possible Errors

AC2	Mnemonic	Meaning
1	ERFNM	Illegal filename.
12	ERDLE	Directory does not exist.
27	ERSPC	Out of disk space.
36	ERDNM	Device or directory not in system.
51	ERNMD	Attempt to initialize too many directories at one time (not enough DCBs specified during system generation).
52	ERIDS	Illegal directory specifier.
53	ERDSN	Directory specifier unknown.
57	ERLDE	Link depth exceeded.
74	ERMPR	Address outside address space.
101	ERDTO	Disk timeout occurred.
112	EROVF	System stack overflow due to excessive number of chained directory specifiers.
121	ERFMT	Disk format error. Try to dump the disk and run DKINIT on it.
122	ERBAD	Disk has invalid bad block table. Dump the disk and run DKINIT on it.

.RLSE

Release a directory or device

This command dissociates a directory or device from the system and prevents further I/O with it. A removable disk should always be released via the CLI's RELEASE command or .RLSE before removing it from the unit. All files within a directory must be closed before releasing it. Release of a master directory releases all directories. The master directory is the directory that holds the current RDOS system; its name is returned by system call .MDIR or the CLI's MDIR command.

Required Input

AC0 - Byte pointer to a directory or device specifier.

Format

```
.SYSTM
.RLSE
error return
normal return
```

Possible Errors

AC2	Mnemonic	Meaning
1	ERFNM	Illegal filename.
31	ERSEL	Unit improperly selected.
36	ERDNM	Device not in system.
56	ERDIU	Directory in use.
66	ERDNI	Directory not initialized.
74	ERMPR	Address outside address space.
101	ERDTO	Disk timeout occurred.
114	ERNIR	Attempted release of a tape unit containing an open file.

.GDIR

Get current directory name

This call returns the name of the current directory or device, for example, DP0. This name is followed by a null; it does not include the names of superior directories or colon specifiers. In the case of current directory DP0F:PART2:DIR1, for example, it would return DIR1.

Required Input

AC0 - Byte pointer to 13_8 byte area to receive the current directory or device name.

Format

```
.SYSTM
.GDIR
error return
normal return
```

The first 12_8 bytes will contain the name, with trailing nulls if necessary; byte 13_8 will contain a null terminator.

Possible Errors

AC2	Mnemonic	Meaning
33	ERRD	Attempt to read into system area.
74	ERMPR	Address outside address space.

.CDIR

Create a subdirectory

This call creates an entry for a subdirectory name in the current partition's system directory (SYS.DR). The subdirectory automatically receives the extension .DR.

Required Input

AC0 - Byte pointer to the directory name (directory specifiers permitted).

Format

```
.SYSTM
.CDIR
error return
normal return
```

Possible Errors

AC2	Mnemonic	Meaning
1	ERFNM	Illegal directory name.
11	ERCRE	Attempt to create an existent directory.
53	ERDSN	Directory specifier unknown.
55	ERDDE	Attempt to create a subdirectory within a subdirectory.
57	ERLDE	Link depth exceeded.
66	ERDNI	Directory not initialized.
74	ERMPR	Address outside address space.
101	ERDTO	Disk timeout occurred.

.CPART

Create a secondary partition

This command creates an entry for a secondary partition name in the current SYS.DR. The secondary partition automatically receives the extension .DR.

Required Input

AC0 - Byte pointer to secondary partition name.

AC1 - Number of contiguous disk blocks in secondary partition. (The minimum is 60_8.) RDOS allocates disk blocks in integer multiples of 20_8; if your number is not an integer multiple of 20_8, the system will truncate it to the lower multiple.

Format

```
.SYSTM
.CPART
error return
normal return
```

Possible Errors

AC2	Mnemonic	Meaning
1	ERFNM	Illegal secondary partition name.
11	ERCRE	Attempt to create an existing secondary partition.
46	ERICB	Insufficient number of free, contiguous disk blocks available.
53	ERDSN	Directory specifier unknown.
54	ERD2S	Partition too small (must have at least 60_8 blocks).
55	ERDDE	Attempt to create a secondary partition within a secondary partition, that is, a tertiary partition.
57	ERLDE	Link depth exceeded.
66	ERDNI	Directory not initialized.
74	ERMPR	Address outside address space.
101	ERDTO	Disk timeout occurred.

.EQIV

Assign temporary name to disk or tape unit

This command assigns a temporary name to a disk or tape unit, permitting unit independence during the execution of your program. Thus, you might write all magnetic tape references in a program as MTAPE and, at runtime, use the .EQUIV command to assign the name MTAPE to a specific device such as MT6. This command must be issued before initializing the device under its new name. The master device cannot be assigned a temporary name.

A device keeps a temporary name until released, then reverts to its original specifier. You can then assign another, temporary name before initialization, if desired.

Required Input

AC0 - Byte pointer to current global specifer name.

AC1 - Byte pointer to temporary name.

Format

```
.SYSTM
.EQIV
error return
normal return
```

Possible Errors

AC2	Mnemonic	Meaning
53	ERDSN	Directory specifier unknown.
56	ERDIU	Device in use, that is, already initialized.
74	ERMPR	Address outside address space.

.GSYS

Get the current operating system name

This call returns the name of the currently executing operating system, its .SV extension, and a null terminator.

Required Input

AC0 - Byte pointer to 15_8 byte area.

Format

```
.SYSTM
.GSYS
error return
normal return
```

The first 12_8 bytes contain the name, with trailing nulls if necessary; bytes 13_8 and 14_8 contain SV, and byte 15_8 contains a null terminator.

Possible Errors

AC2	Mnemonic	Meaning
33	ERRD	Attempt to read or write into system area.
74	ERMPR	Address outside address space.

.MDIR

Get the name of the master directory

Because you can bootstrap an RDOS system in a secondary partition, the master directory may not have an obvious disk name like DP0. The .MDIR command returns the name of the master directory.

Required Input

AC0 - Byte pointer to 13_8 byte area to receive the directory name.

Format

```
.SYSTM
.MDIR
error return
normal return
```

The first 12_8 bytes contains the name, with trailing nulls if necessary; byte 13 contains a null terminator.

Possible Errors

AC2	Mnemonic	Meaning
33	ERRD	Attempt to read or write into system area.
74	ERMPR	Address outside address space.

File Maintenance Commands

The commands described in this section relate to individual files; they enable you to create, delete, set position, and check the status of files. In order of discussion, the file maintenance commands are:

.CCONT	Create a contiguous file with data words zeroed
.CONN	Create a contiguous file with no data words zeroed
.CRAND	Create a random file
.CREAT	Create a sequential file
.DELET	Delete a file
.RENAM	Rename a file
.GPOS	Get the current file pointer
.SPOS	Set the current file pointer
.STAT	Get a file's status
.RSTAT	Get a link entry's resolution file status
.CHSTS	Get a channel's file information
.UPDAT	Update an open file's size information

Each file maintenance command requires that you specify the filename(s) by means of a byte pointer to it. Bits 0—14 of the pointer contain the word address that holds or will receive the first byte. Bit 15 indicates which half: 0 is left, 1 is right. To specify an extension, separate it from the filename with a period. In the following example, the word at location BTPR contains a byte pointer to a properly specified file name, MYFILE.SR.

```
        .TXTM 1
        .
        .
        .
BPTR:   .+1*2
        .TXT "MYFILE.SR"
```

Filenames may include directory specifiers. If you attempt to create a file with the same name as a device in the current system (eg, $LPT), the system treats the command as a no-op and takes the normal return.

.CCONT

Create a contiguously organized file with all data words zeroed

This call creates a contiguously organized file with all data words initialized to zero. If the file's name exists as a link entry and if no resolution file exists for it, RDOS creates a contiguous resolution file.

Required Input

AC0 - Byte pointer to the filename.

AC1 - Number of disk blocks in the file.

Format

```
.SYSTM
.CCONT
error return
normal return
```

Possible Errors

AC2	Mnemonic	Meaning
1	ERFNM	Illegal filename.
11	ERCRE	File already exists.
27	ERSPC	Insufficient disk space to create a .SYS.DR for this file.
46	ERICB	Insufficient number of free, contiguous disk blocks available to create the file.
53	ERDSN	Directory specifier unknown.
57	ERLDE	Link depth exceeded.
66	ERDNI	Directory not initialized.
74	ERMPR	Address outside address space.
101	ERDTO	Disk timeout occurred.
124	ERZCB	Attempt to create a zero length, contiguous file.

.CONN

Create a contiguously organized file with data words zeroed

This command creates a contiguously organized file; it is faster than system call .CCONT because it does not require RDOS to zero the data words. If the file's name exists as a link entry and if no resolution file exists for it, RDOS creates a contiguous, resolution file.

Required Input

AC0 - Byte pointer to filename.

AC1 - Number of disk blocks in the file.

Format

```
.SYSTM
.CONN
error return
normal return
```

Possible Errors

AC2	Mnemonic	Meaning
1	ERFNM	Illegal filename.
11	ERCRE	File already exists.
27	ERSPC	Insufficient disk space to create a SYS.DR entry for this file.
46	ERICB	Insufficient number of free, contiguous disk blocks available to create the file.
53	ERDSN	Directory specifier unknown.
57	ERLDE	Link depth exceeded.
66	ERDNI	Directory not initialized.
74	ERMPR	Address outside address space.
101	ERDTO	Disk timeout occurred.
124	ERCZB	Attempt to create a zero length, contiguous file.

.CRAND

Create a randomly organized file

This command makes an entry for the filename of a randomly organized file in the system directory (SYS.DR), and assigns the first index block to the file. If the file's name exists as a link entry and if no resolution file exists, RDOS creates a random, resolution file.

Required Input

AC0 - Byte pointer to the filename.

Format

```
.SYSTM
.CRAND
error return
normal return
```

Possible Errors

AC2	Mnemonic	Meaning
1	ERFNM	Illegal filename.
11	ERCRE	File already exists.
27	ERSPC	Insufficient disk space to create the file.
53	ERDSN	Directory specifier unknown.
57	ERLDE	Link depth exceeded.
66	ERDNI	Directory not initialized.
74	ERMPR	Address outside address space.
100	ERMDE	Error detected in MAP.DR of non-master device.
101	ERDTO	Disk timeout occurred.

.CREAT

Create a sequentially organized file

This call creates an entry in the system directory (SYS.DR) for the filename of a sequentially organized file, and assigns the first index block to it. If the file's name exists as a link entry and if no resolution file exists, RDOS creates a sequential resolution file.

Required Input

AC0 - Byte pointer to the filename

Format

```
.SYSTM
.CREAT
error return
normal return
```

Possible Errors

AC2	Mnemonic	Meaning
1	ERFNM	Illegal filename.
11	ERCRE	File already exists.
27	ERSPC	Insufficient disk space to create the file.
53	ERDSN	Directory specifier unknown.
57	ERLDE	Link depth exceeded.
66	ERDNI	Directory not initialized.
74	ERMPR	Address outside address space.
101	ERDTO	Disk timeout occurred.

.DELET

Delete a file

Use this command to delete a file and its entry in the system directory. Do not apply it to link entry names, however, or the resolution file will be deleted unless (1) the link access or resolution entry attribute words contain the permanent attribute (in which case RDOS returns error ERDE1); or (2) a resolution file does not exist (ERDLE is returned).

Required Input

AC0 - Byte pointer to filename.

Format

```
.SYSTM
.DELET
error return
normal return
```

Possible Errors

AC2	Mnemonic	Meaning
1	ERFNM	Illegal filename.
12	ERDLE	File does not exist.
13	ERDE1	File is permanent.
53	ERDSN	Directory specifier unknown.
56	ERDIU	Directory in use.
57	ERLDE	Link depth exceeded.
60	ERFIU	File in use.
66	ERDNI	Directory not initialized.
74	ERMPR	Address outside address space.
100	ERMDE	Error detected in MAP.DR of non-master device.
101	ERDTO	Disk timeout occurred.
102	ERENA	Link access not allowed (N attribute).

.RENAM

Rename a file

This call renames a file. It can be applied to a file in a different directory as long as you use the same directory specifier in both the current and new names.

Required Input

AC0 - Byte pointer to the current filename.

AC1 - Byte pointer to the new filename.

Format

```
.SYSTM
.RENAM
error return
normal return
```

After a normal return, the original name no longer exists in the system directory.

Possible Errors

AC2	Mnemonic	Meaning
1	ERFNM	Illegal filename.
11	ERCRE	Attempt to create an existent name (AC1).
12	ERDLE	Attempt to rename a nonexistent file (AC0).
13	ERDE1	Attempt to rename a permanent file (AC0).
35	ERDIR	Files specified in different directories.
53	ERDSN	Directory specifier unknown.
60	ERFIU	File in use.
66	ERDNI	Directory not initialized.
74	ERMPR	Address outside address space.
101	ERDTO	Disk timeout occured.

.STAT and .RSTAT

Get a file's current directory status

These system calls obtain a copy of a file's current directory status. Both calls write a copy of the 22_8-word UFD, as it exists on disk, into an area that you specify. The resulting information can then be accessed via the displacements defined in Table 3.4. When this information pertains to an open file, the result is a "snapshot" of the UFD as it existed on disk at the time of the most recent .CLOSE or .UPDAT.

Use system call .STAT to return the UFD of a file, and .RSTAT to find the UFD of a link's resolution file. Both calls have the same effect on a nonlink file. If you issue .STAT to a link entry, RDOS returns the link's UFD. In a link UFD, words 7 and 14 octal have mnemonics UFLAD and UFLAN while words 7—13 and 14—21 contain the link's alternate directory specifier and alias (if any), respectively.

Offset or Displacement	Mnemonic	Content
00000-00004	UFTFN	Filename (ASCII file number for open tape file)
000005	UFTEX	Extension
000006	UFTAT	File attributes
000007	UFTLK	Link access attributes
000010	UFTBK	Number of the last block in the file (ie, block count —1)
000011	UFTBC	Number of bytes in the last block
000012	UFTAD	Starting logical block address of the file (the random file index for random files)
000013	UFTAC	Year/day last accessed
000014	UFTYD	Year/day created, updated, or closed after write
000015	UFTHM	Hour and minute the file was created, updated, or closed after write
000016	UFTP1	UFD temporary
000017	UFTP2	Number of data words on a disk block
000020	UFTUC	User count (1B0 = .EOPEN, .APPEND, .TOPEN; 1B1 = .OPEN)
000021	UDTDL	DCT link, where bits 10—16 contain device code of device that holds file; left byte is unused except for large disks, for which bits 0—2 contains the high order of the disk address.

Table 3.4 UFD template with displacement mnemonics

Required Input

AC0 - Byte pointer to filename string

AC1 - Starting address of 22_8 word UFD data area.

Format

.SYSTM
.STAT *or* .RSTAT
error return
normal return

Possible Errors

AC2	Mnemonic	Meaning
1	ERFNM	Illegal filename.
12	ERDLE	File does not exist.
33	ERRD	Attempt to read or write into system file space.
36	ERDNM	Device not in system.
53	ERDSN	Directory specifier unknown.
57	ERLDE	Link depth exceeded (.RSTAT only).
66	ERDNI	Directory not initialized.
74	ERMPR	Address outside address space.
101	ERDTO	Device timeout.

.CHSTS

Get the file directory information for a channel

This command returns a copy of current directory status information for the file that is currently open on a specified channel. RDOS returns directory status information as a copy of the 22_8-word UFD, except that it reports status as of the last system—not user—file I/O for this channel. Thus, .CHSTS would return the status after a .WRL, while .STAT or .RSTAT would return the status, on disk, as of the last update or close.

Required Input

AC0 - Starting address of data area. This area must be at least 22_8 words long.

Format

```
.SYSTM
.CHSTS n
error return
normal return
```

Variable n is the file's channel number.

Possible Errors

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
15	ERFOP	No file opened on the given channel.
33	ERRD	Attempt to read into system area.
75	ERMPR	Address outside address space.
101	ERDTO	Disk timeout occurred.

.UPDAT

Update the current file size

This system call allows you to update the size information in a file's UFD while the file is open. The UFD contains a file's size, date of creation, attributes, and other information. In particular, this call updates information in UFTBK and UFTBC in the disk UFD for the file opened on a specified channel, and it writes all modified system buffers not in use to ensure that the file contains all information written to it by your program.

The .UPDAT command is especially useful when a file is open for a long time. Any file that is open during a system failure may contain inaccurate size information in its UFD, preventing you from reading new data. By updating the file frequently, you keep its UFD current and minimize the amount of data that could be lost.

Format

```
.SYSTM
.UPDAT n
error return
normal return
```

Variable n is the file's channel number.

Possible Errors

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
15	ERFOP	File not opened.
101	ERDTO	Disk timeout occurred.

File Attribute Commands

File attribute commands allow you to check or change the current attributes of a file. They can also be used to check device characteristics. The bit settings of AC0 determine the file attributes, while AC1 contains device characteristics of the file. In order of discussion, the file attribute commands are:

.CHATR	Change the attributes of the file opened on channel n.
.GTATR	Get the attributes or characteristics of the file opened on channel n.

Note that these calls work only on an open file. Link commands are discussed in the next section.

.CHATR

Change file attributes

This command changes the access attributes of an open file—or the resolution entry attributes, as viewed from a link entry—according to the contents of AC0.

When you create a file, it has no attributes. If a link user, or a user who has opened via system call .ROPEN, issues the .CHATR command, RDOS temporarily changes his copy of the file attributes until he closes the file; meanwhile, the true resolution entry attributes persist. You must open a file before changing its attributes.

Note that RDOS provides two special attribute bits that can be used to define unique, file access specifications.

Format

```
.SYSTM
.CHATR n
error return
normal return
```

Variable n is the file's channel number.

Required Input

AC0 - An attribute word that contains bits set according to the desired attributes. Set the contents of AC0 according to the bit/attribute relationships show in Table 3.5.

Bit	Symbolic Attribute	Mnemonic	Meaning
1B0	R	ATRP	Read-protected file; cannot be read
1B0	A	ATCHA	Attribute-protected file; no attribute can be changed after you set this bit
1B2	S	ATSAV	Save file (core image file)
1B7	N	ATNRS	No link resolution allowed
1B9	?	ATUS1	First user-definable attribute for the file
1B10	&	ATUS2	Second user-definable attribute for the file
1B14	P	ATPER	Permanent file; cannot be deleted or renamed
1B15	W	ATWP	Write-protected file; cannot be written

Table 3.5 Bit—attribute relationships

Table 3.6 lists the disk file characteristics that RDOS assigns when you create a file. These characteristics cannot be changed by the user.

Bit	Characteristic	Mnemonic	Meaning
1B3	L	ATLNK	Link entry
1B4	T	ATPAR	Disk partition
1B5	Y	ATDIR	Subdirectory
1B6	—	ATRES	Link resolution file (temporary); other file attributes persist for the duration of the open
1B12	C	ATCON	Contiguous file
1B13	D	ATRAN	Random file

Table 3.6 Disk file characteristics assigned by RDOS

Possible Errors

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
14	ERCHA	Illegal attempt to change file attributes (file has A attribute).
15	ERFOP	No file open on this channel.
101	ERDTO	Disk timeout occurred.

.GTATR

Get the file attributes and characteristics

This command obtains the attributes or device characteristics of a file.

Format

```
.SYSTM
.GTATR n
error return
normal return
```

Variable n is the file's channel number. When RDOS returns, AC0 will contain the file attributes. (Table 3.5 described the bit positions that specify attributes.) AC1 will contain the device characteristics of the file. These characteristics pertain to files on reserved devices such as $LPT. They do not reflect the characteristic disable mask supplied when the file was opened. Table 3.7 lists bits and their associated characteristics to aid you in interpreting the bit configuration returned in AC1.

Possible Errors

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
15	ERFOP	Attempt to get attributes of an unopened file.
101	ERDTO	Disk timeout occurred.

AC2	Mnemonic	Meaning
1B0	DCSPC	When file is a spoolable device: spooling enabled (disabled if 0B0)
1B0	DCDIO	When file an MCA link: protocol is suspended on transmit
1B1	DCC80	80-column device
1B2	DCLTU	Device changes lower-case ASCII to upper-case
1B3	DCFFO	Device requiring form feeds on opening
1B4	DCFWD	Full-word device (reads or writes more than a byte)
1B5	DCSPO	Spoolable device
1B6	DCLAC	Output device requiring line feeds after carriage returns
1B7	DCPCK	Input device requiring a parity check; output device requiring parity to be computed
1B8	DCRAT	Output device requiring a rubout after every tab
1B9	DCNAF	Output device requiring nulls after every form feed
1B10	DCKEY	CTRL Z (end-of-file), backslash (line delete), and rubout (character delete) are disabled for this keyboard input device
1B11	DCTO	Teleteypewriter output device or equal leader and trailer $TTP and $PTP
1B12	DCCNF	Output device without form-feed hardware
1B13	DCIDI	Input device requiring operator intervention
1B14	DCCGN	Output device without tabbing hardware
1B15	DCCPO	When file is $TTR/$TTP: output device requiring leader and trailer
1B15	DCSTO	When file is MCA line: user-specified MCA transmitter timeout
1B15	DCNI	When file is MUX line: no CTRL-A or CTRL-C interrupts from this line
1B15	DCSTB	When file is $CDR: trailing blanks are suppressed

Table 3.7 Bits and associated device characteristics

Link Commands

As described in Chapter 2, RDOS permits you to link files in one directory to files in another. Either directory can be a primary partition, secondary partition, or subdirectory. In order of discussion, the link commands are:

.LINK	Create a link entry.
.UNLK	Delete a link entry.
.CHLAT	Change the link access attributes of a file.

.LINK

Create a link entry

This system call creates a link entry from the current directory to a file in the same or another directory. The resulting link entry may or may not have the same name as the resolution file; if not, the link entry's name is referred to as an *alias*. Although no attributes restrict a link when you create it, it cannot reach the resolution file without satisfying both the link entry and file access attributes of the resolution entry. Your program can alter the link, but not the file, access rights of any nonlink file by using the system call .CHLAT. The following examples show the relationships between linknames and various resolution filenames, and their meaning to RDOS.

Linkname	*Resolution Filename*	*Meaning to RDOS*
LFE.SV	LFE.SV	Create link entry LFE.SV in the current directory, and link it to resolution file LFE.SV on the current directory's parent partition.
LFE.SV	SAM:LFE.SV	Create link LFE.SV in the current directory, and link it to resolution file LFE.SV in directory SAM.
NLFE.SV	DP1:LFE.SV	Create link NLFE.SV in the current directory, and link it to resolution file LFE.SV in primary partition DP1.

Required Input

AC0 - Byte pointer to link entry name string.

AC1 - Zero if the link and resolution file have the same name, and if the resolution file resides in the parent partition. Byte pointer to the name string if the link entry has an alias or is not on the parent partition. You may omit a directory specifier from the resolution filename if the resolution file resides on the link entry's parent partition.

Format

```
.SYSTM
.LINK
error return
normal return
```

Possible Errors

AC2	Mnemonic	Meaning
1	ERFNM	Illegal filename.
11	ERCRE	Link entry name already exists.
27	ERSPC	Insufficient disk space to create SYS.DR entry.
53	ERDSN	Directory specifier unknown.
66	ERDNI	Directory not initialized.
74	ERMPR	Address outside address space.
101	ERDTO	Disk timeout occurred.

.ULNK

Delete a link entry

This command deletes a link entry in the directory to which the link entry name points. This command does not delete other links of the same name in other directories. Before issuing .ULNK, make sure that the link entry you are deleting does not also exist between other links and the resolution entry; otherwise, you will be unable to resolve these more remote links after this deletion.

Required Input

AC0 - Byte pointer to the link entry name string.

Format

```
.SYSTM
.ULNK
error return
normal return
```

Possible Errors

AC2	Mnemonic	Meaning
1	ERFNM	Illegal filename.
12	ERDLE	File does not exist.
53	ERDSN	Directory specifier unknown.
66	ERDNI	Directory not initialized.
74	ERMPR	Address outside address space.
75	ERNLE	Not a link entry.
101	ERDTO	Disk timeout occurred.

.CHLAT

Change link access entry attributes

This command changes the link attributes word of the file opened on a channel, according to the contents of AC0. When you open a file via a link entry, the attributes you see will be a composite of the resolution entry's file attributes and your copy of the link access entry attributes. When you create a file, no link entry access attributes exist. Note that RDOS provides two special attribute bits that can be used to define unique, link access specifications.

Required Input

AC0 - File attributes word (identical to .CHATR)

Format

```
.SYSTM
.CHLAT n
error return
normal return
```

Variable n is the channel number.

Possible Errors

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
14	ERCHA	Resolution entry is attribute-protected (has A attribute).
15	ERFOP	No file is open on this channel.
101	ERDTO	Disk timeout occurred.

Input/Output Commands

This section describes the system calls a program may use to write data to, and read data from, an existing, open file. It begins by describing the five I/O modes available, and goes on to explain the calls that open and close a file. Then the calls used to change position in a file are discussed, followed by descriptions of the different writing and reading calls themselves.

Generally, you can do nothing with a file until you have opened and assigned it a channel number with one of these commands: .OPEN, .EOPEN, .ROPEN,.APPEND, or .MTOPD. Remember, too, that a file may be a device such as $TTI for console input, or a disk file such as MYFILE.SR, which can include a directory specifier (eg, DP1:MYFILE.SR) if you have initialized the directory.

In order of discussion, the file I/O calls are:

.OPEN n	Open a file for I/O on channel n.
.EOPEN n	Open a file for exclusive writing on channel n.
.ROPEN n	Open a file for reading only on channel n.
.APPEND n	Open a file for appending on channel n.
.GCHN	Get the number of a free channel.
.CLOSE n	Close the file on channel n.
.RESET	Close all files.
.GPOS n	Get the position of the file pointer.
.SPOS n	Set the position of the file pointer.
.RDL n	Read an ASCII line from a file.
.WRL n	Write an ASCII line to a file.
.RDS n	Read sequential bytes from a file.
.WRS n	Write sequential bytes to a file.
.RDR n	Read a 64-word record.
.WRR n	Write a 64-word record.
.RDB n	Read a series of disk blocks from or to a file, without a system buffer.
.WRB n	Write a series of disk blocks from or to a file, without a system buffer.
.MTOPD n	Open a magnetic tape file for free-form I/O.
.MTDIO n	Write or read data to or from a magnetic tape file in free form.

If RDOS detects an error when it executes an I/O command, it reattempts the command, if possible, before reporting the error with code ERFIL.

Input/Output Modes

RDOS provides five basic modes for reading and writing files:

line
sequential
random record
direct block
free form (tape)

The line and sequential modes are generally used for ASCII character strings and binary files, respectively.* Random record mode allows you to read or write 64-word records, while direct-block I/O allows you to transfer a contiguous group of disk blocks without a system buffer. Free-form I/O allows you to read or write free form blocks of data to magnetic tape.

Line Mode

In line mode, the system assumes that the data you want to read or write consists of ASCII character strings terminated by a carriage return, form feed, or null character. RDOS processes file data line by line, in sequence, from the beginning of the file to its end.

In line mode, the system handles all device-dependent editing at the device driver level. Furthermore, reading and writing never require byte counts, since reading continues until RDOS reads a terminator and writing proceeds until the user writes one. The line mode commands include .RDL (read a line) and .WRL (write a line).

Sequential Mode

In unedited sequential mode, RDOS transmits data exactly as it is read from or written to the file or device. This mode is required for the processing of sequential, binary files. To use sequential mode, your program must specify the byte count necessary to satisfy a read or write request. The sequential mode commands are .RSD (read sequential) and .WRS (write sequential).

*The RDOS system library contains a module called the Buffer I/O Package that speeds up line and sequential mode operations. The module is described in application notes.

In line or sequential modes, your position within a file is always the position at the end of your last .SPOS, line mode, or sequential mode command. The first read or write occurs at the beginning of the file unless your program opened this file for appending.

Random Record Mode

Random record mode permits random access to fixed-length records within random or contiguous disk files. The fixed length of a random record is 100_8 words. The system calls for this mode are .RDR (read a record) and .WRR (write a record).

Direct Block Mode

Direct block I/O allows you to transfer a continuous group of blocks in a random or contiguous file without using a system buffer. RDOS uses sequential memory locations for this purpose, and transfers only 512-byte blocks of data between memory and disk. Relative block numbers must be transferred in an unbroken series. Thus, you may process the third, fourth, and fifth blocks in a file in a single call, but not the third, fifth, and sixth blocks. Direct block I/O can be executed with the system calls .RDB (read a series of blocks) and .WRB (write a series of blocks). Note that window mapping, which permits extended, direct block I/O, can be employed in a mapped system. In this mode, your program can transfer disk blocks to and from extended address space via the system calls .ERDB and .EWRB, as described in Chapter 4.

Free Form Mode

Finally, free form I/O permits you to read or write free-form blocks of data to magnetic tape. In this mode, you can read or write from two to 4096-word data records; space forward or backward through one to 4096 data records or to the start of a new data file; and read the transport status word. To use free-form I/O, a file must be opened via the .MTOPD command and its operation directed via call .MTDIO. The latter cannot be mixed with the .WRL or .WRS commands on the same tape drive.

.OPEN

Open a file

Before a program can issue other I/O commands, it must associate a file to an RDOS channel number. The .OPEN command associates a file with a channel number and makes the file available to any user for reading and writing. The command does not guarantee exclusive use of the file; others may also have opened the file via .OPEN and modified its contents. A file must be closed before it can be deleted or renamed.

There is no RDOS command that reduces the size of a file. Thus, files never shrink but maintain space for all material written to them by any user. To remove redundant or useless material from a file, edit it with the Text Editor utility; or, using file position and system write calls, overwrite the useless data with nulls or new material.

Required Input

AC0 - Byte pointer to the filename.

AC1 - Characteristic disable mask (except for MCA lines). For every bit set in the mask word, RDOS disables the corresponding device characteristic for the duration of the .OPEN. (See also Table 3.7 under "File Attribute Commands" earlier in this chapter.) For example, if you want to read an ASCII tape without parity checking from the paper tape reader, you can disable checking by the following:

```
        .
        LDA 0,READR
        LDA 1,MASK
        .SYSTM
        .OPEN 3
        .
        .
        .
READR:  +1*2
        .TXT "$PTR"
MASK:   DCPCK           ;DISABLE PARITY
                        ;CHECKING.
```

RDOS normally restricts console output to 80 columns. If your terminal is a DASHER®, you can instruct RDOS to print the full 132 columns by opening $TTO ($TTO1) with disable bit DCC80 set, for example:

```
.
LDA 0, NTTO
LDA 1, DMASK
.SYSTM
.OPEN n
.
.
NTTO:   +1*2
        .TXT            "$TTO"
DMASK:    DCC80
```

To use system mnemonics like mask and error words, your program should be assembled with the macroassembler utility. The assembler's symbol table file must include PARU.SR. In general, you will want to preserve all device characteristics defined by the system. To do so, insert a SUB 1, 1 instruction before the .OPEN call.

To open an MCA line for transmit, you must specify a transmit timeout period, rather than a mask, in AC1. Set AC1 to 0 to specify the default timeout period of 655 seconds. For a shorter timeout period, set AC1 to 1, specifying the actual timeout period in the write-sequential call, .WRS. Pass 0 in AC1 to open an MCA line for receiving.

Format

```
.SYSTM
.OPEN n
error return
normal return
```

Variable n becomes the channel number of the file until it is closed.

In a multitask environment, a task that opens a disk file previously opened by another task cannot read past the end-of-file point that existed when it opened the file, even if a previous task extends the file. Also note that the .OPEN system call executes in two parts to allow efficient performance in a multitasking environment. If .OPEN is immediately followed by a .CLOSE or a .RESET, timing problems may arise. If a timing problem exists, insert the command .DELAY ahead of the .CLOSE or .RESET calls. One further point of interest to multitask developers is that RDOS interleaves line printer output if multiple tasks in the same program write to the printer. If an opened file requires leader, RDOS outputs it on the .OPEN. If an opened file requires intervention, RDOS displays the message LOAD *filename*, STRIKE ANY KEY.

Possible Errors

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
1	ERFNM	Illegal filename.
12	ERDLE	File does not exist.
21	ERUFT	Attempt to use channel already in use.
27	ERSPC	File space exhausted.
31	ERSEL	Unit improperly selected.
36	ERDNM	Device not in system.
53	ERDSN	Directory specifier unknown.
57	ERLDE	Link depth exceeded.
60	ERFIU	File opened for exclusive use (.EOPEN).
66	ERDNI	Directory not initialized.
74	ERMPR	Address outside address space.
101	ERDTO	Disk timeout occurred.
102	ERENA	No linking allowed (N attribute).
111	ERDOP	Attempted open of an open tape file.

.EOPEN

Open a file for exclusive write access

This command gives you exclusive write access to a file. Thus, only you can modify a file when you open it via .EOPEN, although other users may gain read access to it via the .ROPEN command. RDOS cannot exclusively open peripheral filenames such as $LPT, although an attempt to do so will not result in an error. Multiple .EOPENs have exactly the same effect as multiple .OPENs.

Required Input

AC0 - Byte pointer to filename.

AC1 - Characteristic disable mask.

Format

```
.SYSTM
.EOPEN n
error return
normal return
```

Variable n is the file's channel number.

Possible Errors

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
1	ERFNM	Illegal filename.
12	ERDLE	File does not exist.
21	ERUFT	Attempt to use channel already in use.
31	ERSEL	Unit improperly selected.
36	ERDNM	Device not in system.
53	ERDSN	Directory specifier unknown.
57	ERLDE	Link depth exceeded.
60	ERFIU	File already opened for writing.
66	ERDNI	Directory not initialized.
74	ERMPR	Address outside address space.
101	ERDTO	Disk timeout occurred.
102	ERENA	No linking allowed (N attribute).
111	ERDOP	Attempt to open a file that is already open.

.ROPEN

Open a file for reading only

This system call opens a file for reading only. A program may gain read-only access to a file that is currently open as a result of an .EOPEN, .OPEN, or another .ROPEN command. Thus, several users may access a file for reading only while one of them has write-access to it. All users must have closed the file before anyone can delete or rename it.

Required Input

AC0 - Byte pointer to filename.

AC1 - Characteristic disable mask.

Format

```
.SYSTM
.ROPEN n
error return
normal return
```

Variable n is the file's channel number.

Possible Errors

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
1	ERFNM	Illegal filename
12	ERDLE	File does not exist.
21	ERUFT	Attempt to use channel already in use.
32	ERSEL	Unit improperly selected.
36	ERDNM	Device not in system.
53	ERDSN	Directory specifier unknown.
57	ERLDE	Link depth exceeded.
66	ERDNI	Directory not initialized.
74	ERMPR	Address outside address space.
101	ERDTO	Disk timeout occurred.
102	ERENA	No linking allowed (N attribute).
111	ERDOP	Attempt to open an open tape file.

.APPEND

Open a file for appending

This system call is identical to the .EOPEN command, except that it opens a file specifically for appending. If your program attempts to read such a file, RDOS returns error code EREOF (end-of-file), because the file pointer is positioned after the last byte.

RDOS opens a disk file and appends whatever you write to it. On a magnetic tape device, RDOS opens the tape file, reads to the end-of-file (EOF), and then writes from that point. RDOS opens the line printer without a form feed.

Note that if you plan a BATCH environment in which a program outputs to, say, file SYSOUT, that file must be opened for appending, not simply opened.

Required Input

AC0 - Byte pointer to filename.

AC1 - Device characteristic disable mask.

Format

```
.SYSTM
.APPEND n
error return
normal return
```

Variable n is the file's channel number.

Possible Errors

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
1	ERFNM	Illegal filename.
3	ERICD	Illegal command for device.
12	ERDLE	File does not exist.
21	ERUFT	Attempt to use channel already in use.
31	ERSEL	Unit improperly selected.
36	ERDNM	Device not in system.
53	ERDSN	Directory specifier unknown.
57	ERLDE	Link depth exceeded.
60	ERFIU	File in use.
66	ERDNI	Directory not initialized.
74	ERMPR	Address outside address space.
101	ERDTO	Disk timeout occurred.
102	ERENA	No linking allowed (N attribute).
111	ERDOP	Attempt to open a file that is already open.

.GCHN

Get the number of a free channel

This system call returns the number of a free channel in AC2. Your program can then use AC2 to open a file via one of the open calls. The command does not open a file on a free channel, but merely indicates a channel that is free at the moment. Occasionally, in a multitask environment, you will find that the channel indicated by .GCHN is no longer free when you issue your open. In this case, the system returns error ERUFT, indicating that you should reissue .GCHN to discover another free channel.

Format

```
.SYSTM
.GCHN
error return
normal return
```

Upon a normal return, RDOS returns the free channel number in AC2.

Possible Errors

Only one error is possible: its mnemonic is ERUFT, its error code (returned in AC2) is 21, and it occurs when no channels are free.

.CLOSE

Close a file

A file must be closed after use in order to update its UFD in the system directory, to delete it, or to release its directory or device. When you close a file, its channel number becomes available for other I/O. The system calls .RTN, .ERTN, .BREAK, and .RESET automatically close all channels. In a multitask environment, it is imperative that all read and write commands to the same channel be allowed to complete their execution before issuing the .CLOSE command. The only exception is when using .CLOSE to abort I/O operations on an MCA or QTY device.

Format

```
.SYSTM
.CLOSE n
error return
normal return
```

Variable n is the channel number.

Possible Errors

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
15	ERFOP	Attempt to close a channel not in use.
101	ERDTO	Disk timeout occurred.

.RESET

Close all files

This command closes all open files after writing any partially filled system buffers. The .RESET command can be issued in a multitask environment only when no other task is using a channel.

Format

```
.SYSTM
.RESET
error return
normal return
```

Possible Errors

Only one error is possible. Its mnemonic is ERDTO, its error code (returned in AC2) is 101, and it results when a disk timeout has occurred.

.GPOS

Get the current file pointer

This command is used to determine the next character position within a file where program writes or reads will occur. RDOS indicates a relative character position within a file by a double-precision byte pointer. This two-word byte pointer contains the high-order portion of the byte address in AC0 and the low-order portion of the byte address in AC1. Bit 15 of the second word indicates the byte selection (left or right), as shown in Figure 3.1.

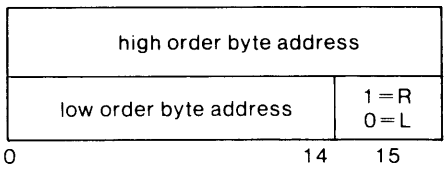


Figure 3.1 Double-precision byte pointer DG-25452

Format

```
.SYSTM
.GPOS n
error return
normal return
```

Variable n is the file's channel number. RDOS returns the pointer position in AC0 and AC1 as just described. It returns zero if you open a nondisk file on channel n.

Possible Errors

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
15	ERFOP	No file is open on this channel.

.SPOS

Set the current file pointer

This system call sets the current, system file pointer to a new character position in preparation for future writing or reading. It enables you to access characters and lines randomly within any block of a given file, and allows you to read a character after writing or rewriting it by simply backing up the pointer to its previous position.

RDOS indicates the relative character position within a file by the double-precision byte-pointer illustrated in Figure 3.1. If you set the file pointer beyond the end of a file, RDOS automatically extends its length. If the file is contiguous, that is, cannot be extended, RDOS takes the error return and passes ERSCP in AC2. Only position 0, the file starting location, can be specified on magnetic tape.

Required Input

AC0 - High-order portion of byte pointer.

AC1 - Low-order portion of byte pointer.

Format

```
.SYSTM
.SPOS n
error return
normal return
```

Variable n is the file's channel number.

Possible Errors

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
15	ERFOP	Attempt to reference an unopened file.
64	ERSCP	File position error.

.RDL

Read a line

This command reads an ASCII line from a file to the area of user memory that you specify. This are should be 133 decimal bytes in length, and AC0 must contain a byte pointer to the starting byte address within user memory into which RDOS will read the line.

An .RDL operation terminates normally after RDOS has read either a carriage return, form feed, or null and transmitted it to your program. The system stops reading and takes the error return if it transmits 133 characters without detecting a terminator or upon discovering a parity error or end of file. The operation also terminates if RDOS reads a preassigned interrupt character from a multiplexed line.

If RDOS is reading from the keyboard ($TTI or $TTI1), its controls work as usual unless you have masked DCKEY in AC1, as discussed under the .GTATR command. Rubout deletes the preceding character, and backslash (SHIFT-L) deletes the preceding line, from the keyboard stream. RDOS echoes all printing characters and ignores line feeds. An end of file is indicated by pressing CTRL-Z. Note that when reading from a multiplexed line, ESC also indicates an end of file.

When the card reader serves as an input device to the .RDL command an end of file must be indicated by punching all rows in column 1, that is, multipunching the characters +, —, and 0 through 9. The Hollerith-to-ASCII translation that occurs during an .RDL operation assumes the keypunch codes shown in Appendix B. The operation terminates the first trailing blank unless your .OPEN command suppressed DCSTB, causing RDOS to transfer all 80 characters. In this case, RDOS appends a carriage return as the 81st character unless your .OPEN command suppressed DCC80, allowing the system to process a maximum of 72 characters. RDOS replaces each illegal character with a backslash.

Note that where card readers are concerned, RDOS ignores all columns following the EOF. The card reader driver permits an unlimited amount of time to elapse until it reads the next card, thereby permitting the operator to correct pick errors or insert new card files. Because the driver employs double buffering, you will lose at least one card image if you close prematurely; a program must therefore wait until RDOS reads the last card or end of file to close $CDR. After closing, the reader can be reopened without loss of data, and reading may continue. When RDOS reads an end-of-file card, it returns a byte count of 0 along with EREOF. If another .RDL command is issued, RDOS reads the next card normally.

For all files and devices, RDOS returns the number of bytes read, including the terminator, in AC1. If the read terminates

because of a parity error, RDOS stores the character having incorrect parity as the last character read and clears the parity bit. The algorithm for computing the byte pointer to the bad character is (AC0) + (AC1) − 1, where (AC0) means the contents of accumulator.

Required Input

AC0 - Byte pointer to receiving buffer.

Format

```
.SYSTM
.RDL n
error return
normal return
```

Variable n is the channel number of the file from which RDOS will read. After a normal return, AC1 contains the number of bytes read.

Possible Errors

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
3	ERICE	Illegal command for device.
6	EREOF	End of file.
7	ERRPR	Attempt to read a read-protected file.
15	ERFOP	Attempt to reference a file not open.
22	ERLLI	Line limit (133 nonterminator characters) exceeded.
24	ERPAR	Parity error; often occurs on tape due to dirty heads.
30	ERFIL	File read error; often signals a bad tape or a tape drive with dirty heads.
33	ERRD	Attempt to read into system area.
34	ERDIO	File accessible by direct block I/O only.
47	ERSIM	Simultaneous reads from the same multiplexor (ALM/QTY) line.
74	ERMPR	On mapped systems only: address outside address space.
101	ERDTO	Disk timeout occurred.
106	ERCLO	Channel closed by another task.

.WRL

Write a line

This command, the counterpart of .RDL, writes an ASCII line to the file open on a specified channel. AC0 must contain a byte pointer to the starting byte address within user memory from which characters will be written. Writing commences at the start of the file unless you have moved the file pointer via the .SPOS command or opened the file via .APPEND.

Normal operation stops when the system detects a null, carriage return, or form feed. Under abnormal circumstances, the system stops writing after it transmits 132 decimal characters without detecting a terminator as the 133rd character.

Upon termination, AC1 contains the number of bytes written from your area of memory to the file. The null terminator does not force a carriage return or line feed. A carriage return generates a line feed on ouput, provided the device characteristics so dictate.

Required Input

AC0 - Byte pointer to starting byte address.

Format

```
.SYSTM
.WRL n
error return
normal return
```

Variable n represents the channel number of a file to which the system will write.

Possible Errors

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
3	ERICD	Illegal command for device.
6	EREOF	End of file when writing to a contiguous file.
10	ERWPR	Attempt to write to a write-protected file.
15	ERFOP	Attempt to write to a file not opened.
22	ERLLI	Line limit (132 characters).
27	ERSPC*	Out of disk space.
34	ERDIO	File accessible by direct block I/O only.
47	ERSIM	Simultaneous writes to the same multiplexor (ALM/QTY) line.
74	ERMPR	Address outside address space.
101	ERDTO	Disk timeout occurred.
103	ERMCA	The MCA receiver on this channel issued no transmit request.
104	ERSRR	MCA transmission terminated by receiver (short receive request).
106	ERCLO	Channel closed by another task.

*If you write to a sequential or random file and get ERSPC, you must delete the file in order to recover the disk space allocated to the file before the error occurred. You must do this even though the CLI LIST command may show a zero file length.

.RDS

Read sequential

In sequential mode, RDOS transmits data exactly as read to or written from a file. This mode must be used for binary data, and is often helpful for MCA transmissions. The command instructs RDOS to read data exactly as it appears in the file, unless it is reading from the system console. In this case, RDOS sets the parity bits to zero. Note that in read sequential mode, the system does not recognize CTRL-Z from the console as an end-of-file character. Upon detecting a legitimate end of file, RDOS returns the partial byte count in AC1.

Where card readers are concerned, RDOS reads the card in image binary, using each of two bytes to read a single column and packing them as shown in Figure 3.2. Each variable d in this figure will be 1 for every punched hole in the column. A byte pair containing the word 100000 signifies an end of card (EOC). Thus, to read two entire 80-column cards one at a time, you would issue two successive .RDS commands for 162 bytes each. If you request only 160 bytes for each read, the second .RDS command returns the first end-of-card word, along with the first 79 columns of the second card.

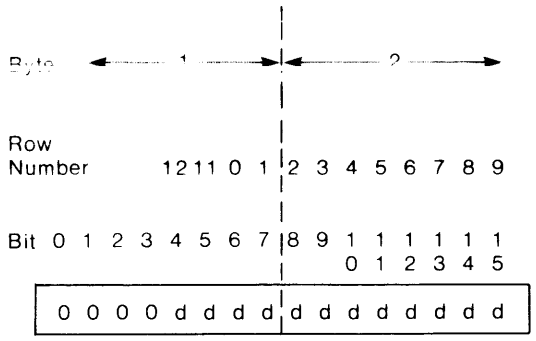


Figure 3.2 Image binary card reading SD-00430A

Required Input

AC0 - Byte pointer to the starting byte address within user memory into which RDOS will read data.

AC1 - Number of bytes to be read.

Format

```
.SYSTM
.RDS n
error return
normal return
```

Variable n is the channel number of a file from which data will be read.

Possible Errors

AC2	Mnemonic	Meaning
0	ERFNO	Illegal character number.
3	ERICD	Illegal command for device.
6	EREOF	End of file.
7	ERRPR	Attempt to read a read-protected file.
15	ERFOP	Attempt to reference a file not open.
24	ERPAR	Parity error on tape, often caused by dirty heads.
30	ERFIL	File read error, often caused by bad tape or dirty heads.
33	ERRD	Attempt to read into system area.
34	ERDIO	File accessible by directory block I/O only.
47	ERSIM	Simultaneous reads from same multiplexed line.
74	ERMPR	Address outside address space.
101	ERDTO	Disk timeout occurred.
103	ERMCA	The MCA transmitter issued no transmit request.
106	ERCLO	Channel closed by another task.

.WRS

Write sequential

This command, the counterpart of .RDS, writes data verbatim from memory to a file. Note that RDOS recognizes no character as an end of file in this mode. The system commences writing at the start of the file unless you have moved the file pointer via the .SPOS command or opened the file via .APPEND.

Required Input

AC0 - Byte pointer to the starting address of the data within user memory.

AC1 - Number of bytes to be written.

Format

```
.SYSTM
.WRS n
error return
normal return
```

Variable n is the channel of a file to which data will be written. To transmit (write) data over an MCA line, you must pass an even byte pointer in AC0 and specify an even byte count in AC1. If you opened this MCA channel and specified a nondefault retry period in AC1, you must also define the length of the timeout period in the left byte of AC2. Each retry takes about 200 milliseconds. Acceptable values for AC2 range from 1 to 377_8. If the left byte of AC2 is 0, RDOS allots the maximum transmit retry period of approximately 655 seconds.

To send an end of file over an MCA line, set AC1 to 0; RDOS disregards the contents of AC0. Chapter 8 describes MCA programming in greater depth.

Possible Errors

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
3	ERICD	Illegal command for device.
6	EREOF	End of file when writing to a contiguous file.
10	ERWPR	Attempt to write to a write-protected file.
15	ERFOP	Attempt to write to a file not open.
27	ERSPC*	Out of disk space.
34	ERDIO	File accessible by direct block I/O only.
47	ERSIM	Simultaneous writes to the same QTY/ALM line.
74	ERMPR	Address outside address space.
101	ERDTO	Disk timeout occurred.
103	ERMCA	The MCA receiver on this channel issued no receive request.
104	ERSRR	MCA transmission terminated by receiver (short receive request).
106	ERCLO	Channel closed by another task.
113	ERNMC	No outstanding receive request.

*If you write to a sequential or random file and get ERSPC, you must delete the file in order to recover the disk space allocated to the file before the error occurred. You must do this even though the CLI LIST command may show a zero file length.

.RDR

Read random record

This system call allows a program to read one 64-word record in either a random or contiguous disk file. Each disk block contains four, 64-word records numbered 0, 1, 2, and 3 in the first block of a file; 4, 5, 6, and 7 in the second block; and so on. These numbers need only be considered when issuing the random record commands. To read or write blocks, that is, four records at a time, you would use system calls .RDB or .WRB, and the commands .RDL, .WRL, .RDS, or .WRS to read or write lines.

Required Input

AC0 - Destination memory address.

AC1 - Record number. (Record numbers start with 0.)

Format

```
.SYSTM
.RDR n
error return
normal return
```

Variable n is the channel number of a file from which data will be read.

Possible Errors

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
3	ERICD	Illegal comand for device.
6	EREOF	Attempt to read past the end of a contiguous file.
7	ERRPR	Attempt to read a read-protected file.
15	ERFOP	No file is open on this channel.
30	ERFIL	File read errors, usually due to bad tape.
33	ERRD	Attempt to read into system area.
34	ERDIO	File accessible by direct block I/O only.
74	ERMPR	Address outside address space.
101	ERDTO	Disk timeout occurred.

.WRR

Write random record

This command writes a 64-word record from memory to a randomly or contiguously organized disk file. RDOS writes 64 words to the record number specified, starting from the address that you pass in AC0.

Required Input

AC0 - Memory address.

AC1 - Destination record number.

Format

```
.SYSTM
.WRR n
error return
normal return
```

Variable n is the channel number of a file to which data will be written.

Possible Errors

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
3	ERICD	Illegal command for device.
6	EREOF	Atempt to write past the end of a contiguous file.
10	ERWPR	Attempt to write to a write-protected file.
15	ERFOP	Attempt to reference a file not opened.
27	ERSPC*	Out of disk space.
34	ERDIO	File accessible by direct block I/O only.
74	ERMPR	Address outside address space.
101	ERDTO	Disk timeout occurred.

*If you write to a sequential or random file and get ERSPC, you must delete the file in order to recover the disk space allocated to the file before the error occurred. You must do this even though the CLI LIST command may show a zero file length.

.RDB or .WRB

Read or write a series of disk file blocks

These system calls, for direct block I/O, are used to transfer blocks to or from random or contigous files. RDOS employs no system buffers for the transfer. Blocks in random and contiguous disk files have a fixed length of 256 decimal words, and are numbered sequentially from 0. Thus, an .RDB command issued for the first block in a file would transfer the 64-word records numbered 1, 2, and 3, as described earlier under system call .RDR.

Required Input

AC0 - Starting memory address for the block transfer.

AC1 - Starting relative block number in the series to be transferred.

AC2 - The left half of AC2 must contain the number of blocks to be transferred. The right half of AC2 must contain the channel number if you specify channel 77.

Format

```
.SYSTM
.RDB or .WRB n
error return
normal return
```

Variable n represents the channel number.

Possible Errors

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
3	ERICD	Illegal command for device.
4	ERSV1	Not a random or contiguous file.
6	EREOF*	End of file.
7	ERRPR	File is read-protected (.RDB).
10	ERWPR	File is write-protected (.WRB).
15	ERFOP	File is not open.
27	ERSPC*	Disk space is exhausted.
30	ERFIL	File read error, usually on magnetic tape due to a bad tape or dirty head.
33	ERRD	Attempt to read into system area (.RDB).
40	EROVA	File not accessible by direct block I/O.
74	ERMPR	On mapped systems only: address outside address space.
101	ERDTO	Disk timeout occurred.

*Upon detection of error EREOF or ERSPC, RDOS returns the code in the right byte of AC2; the left byte contains the partial read or write count.

If you write to a sequential or random file and get ERSPC, you must delete the file in order to recover the disk space allocated to the file before the error occurred. You must do this even though the CLI LIST command may show a zero file length.

.MTOPD

Open a tape unit and file for free format I/O

Before you can read or write in free format on magnetic tape, the device must be opened and associated with a channel. The .MTOPD command performs this function. It is a global system call, allowing access to all files on the specified device after it is issued.

To position a free format tape to a specific file, pass the filename to .MTOPD in the form MT*n*:*m*, where *n* is the drive number, and *m*, the file number. After completing all operations on a tape drive, remember to release it.

Required Input

AC0 - Byte pointer to the magnetic tape file specifier.

AC1 - Characteristic disable mask, as described earlier under system call .GTATR.

Aside from the tape file specifier, these parameters are identical to those for the .OPEN command. To learn more about device characteristics, see the descriptions of the .OPEN and .GTATR commands earlier in this chapter.

Format

```
.SYSTM
.MTOPD n
error return
normal return
```

Variable n represents the channel number.

Possible Errors

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
1	ERFNM	Illegal filename.
3	ERICD	Illegal command for device.
12	ERDLE	File does not exist.
21	ERUFT	Attempt to use a channel already in use.
27	ERSPC	File space exhausted.
31	ERSEL	Unit improperly selected.
36	ERDNM	Device not in system.
53	ERDSN	Directory specifier unknown.
57	ERLDE	Link depth exceeded.
66	ERDNI	Directory not initialized.
74	ERMPR	Address outside address space.
111	ERDOP	Attempted open of an open tape file.

.MTDIO

Perform free format I/O

This command provides a direct interface with magnetic tape units on a machine level. It enables you to read or write data in variable length records of 2 to 4096 words; to space forward or backward from 1 to 4096 data records or to the start of a new data file; and to perform similar, machine-level operations. Before any of these operations can be performed, the tape unit must be opened for free format I/O with system call .MTOPD. For information about the hardware characteristics, see the manual *Peripherals, Programmer's Reference Series* (DGC No. 014-000632).

Required Input

The following input is required to read the device status word.

AC0 - Command word, with bits 1 through 3 set and all other bits 0.

AC2 - Channel number, if equal to 77.

The following input is required for other .MTDIO operations.

AC0 - Memory address for data transfer.

AC1 - Command word, subdivided into three fields:

Bit 0 — Set to 1 for even parity, 0 for odd parity.

Bits 1—3 — Set to one of these seven command codes: 0 for reading words;* 1 for rewinding the tape; 3 for spacing forward over records or over a file of any size up to 4096 words; 4 for spacing backward over records or a file of up to 4096 records in size; 5 for writing words; 6 for writing end of file (odd parity for 9-track, even parity for 7-track); 7 for reading device status word.

Bits 4—15 — Word or record count. If 0 on a space forward or backward command and the file is no more than 4096 words, RDOS positions the tape to the beginning of the next (or previous) file on the tape. If 0 on a read command, RDOS reads words until it encounters either an end of record or 4096 words. If 0 on a write command, the system writes 4096 words.

AC2 - Channel number, if equal to 77.

*When reading a 7-track tape with odd parity, that is, a tape not written on an RDOS system, the controller does not detect the end of file; instead, it reads the first word in the next record as 007417. Thus, RDOS appends the first record of each file after the first to the EOF of the previous file.

Format

```
.SYSTM
.MTDIO n
error return
normal return
```

Variable n represents the channel number.

If no system error is detected during a read status command, RDOS takes the normal return and AC2 contains a device status word with one or more bits set. These bits are shown in Figure 3.3.

When your program issues a read, write, space forward, or space backward command, the command word in AC1 contains the number of words written or read, or the number of records spaced. The system returns a word or record count if it encounters a premature end of file.

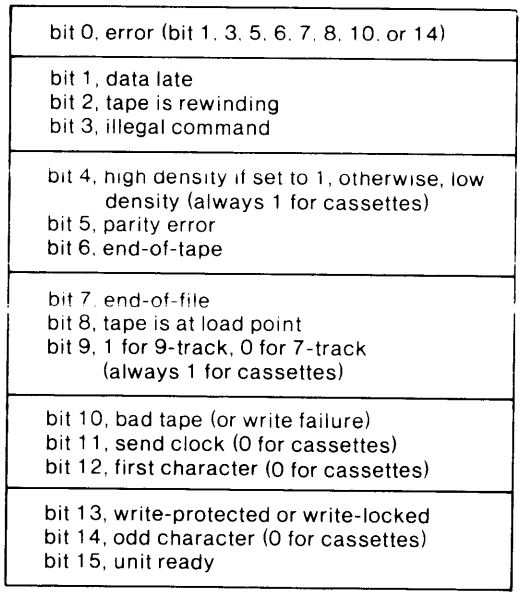


Figure 3.3 MTDIO status word bits SD-00540A

Possible Errors

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
3	ERICD	Illegal command for device (ie, improper open).
15	ERFOP	Attempt to reference a file not opened.
40	EROVA	File not accessible by free form I/O.
74	ERMPR	Address outside address space.

Table 3.8 summarizes the possible returns by .MTDIO and the values passed in AC1 and AC2. RDOS sets bit 0 of TSW (in AC2) when a hardware error occurs, and clears this bit in the event of a system error. The system retries a read operation 10 times before taking the error return. For write errors, it takes the error return after 10 attempts to backspace, erase a length of tape, and write.

COMMAND	RETURN	AC1	AC2
Any .MTDIO command with a system error detected	Error	Same as input	System error code
Rewind	Normal	Original input lost	Transport status word (TSW)
Rewind (tape at load point, etc.)	Error		
Read Status	Normal	Original input lost	TSW
Read Status	Error		TSW
Read, Write Space Forward Space Backward	Normal	Word or record count	TSW
Read, Write, Space Forward, Space Backward	Error (only after 10 retries in read/write		
Write EOF	Error	Original input lost	TSW

Table 3.8 .MTDIO values returned

Console I/O Commands

This section begins by describing the .GCHAR and .PCHAR commands, which transfer single characters between your console and AC0. These calls operate in the manner of a read or write sequential of one character. They do not affect the column counter, nor do they provide special character handling (eg, of carriage returns). Both commands reference $TTI/$TTO or $TTI1/$TTO1; the console is always available to them, and no channel number or open command is required.

Also discussed in this section are the .GCIN and .GCOUT commands, which return the name of the console I/O device.

.GCHAR

Get a character

This command places a character typed on the console in AC0. RDOS right-adjusts the character, without parity in AC0, and clears the left byte of this accumulator. The system does not echo this character on the console. No I/O channel for .GCHAR need be specified to issue the .GCHAR command.

Format

```
.SYSTM
.GHCAR
error return
normal return
```

If the console input buffer does not contain a character, the system waits.

Possible Errors

Only one error is possible as a result of this command. Its mnemonic is ERICD, meaning that the console is not in the system, and RDOS passes code 3 in AC2 when it occurs.

.PCHAR

Put a character

This system call types the character in bits 9 through 15 of AC0 on the console.

Format

```
.SYSTM
.PCHAR
error return
normal return
```

Possible Errors

Only one error is possible as a result of this command. Its mnemonic is ERICD, meaning that the console has not been defined to the system, and RDOS passes error code 3 in AC2 when it occurs.

.GCIN

Get the input console name

This command returns the name of the current console input device: $TTI for the background program, and $TTI1 for the foreground program. The .GCIN command and its counterpart, .GCOUT, are useful in dual-ground systems because they allow each program to select the appropriate console for its ground at runtime.

Required Input

AC0 - Byte pointer to a six-byte area that will receive the console name.

Format

```
.SYSTM
.GCIN
error return
normal return
```

Possible Errors

AC2	Mnemonic	Meaning
33	ERRD	On unmapped systems only: attempt to read into system area.
74	ERMPR	Address outside address space.

.GCOUT

Get the output console name

This command returns the name of the current output console: $TTO for the background program, and $TTO1 for the foreground program.

Required Input

AC0 - Byte pointer to the six-byte area that will receive the console name.

Format

```
.SYSTM
.GCOUT
error return
normal return
```

Possible Errors

AC2	Mnemonic	Meaning
33	ERRD	Attempt to read into system area (unmapped only).
74	ERMPR	Address outside address space.

Memory Allocation Commands

Excluding the Task Scheduler and octal locations 0 through 15, RDOS resides in upper memory and excutes user programs in lower memory. Figure 3.4 diagrams RDOS memory as it exists in unmapped systems.

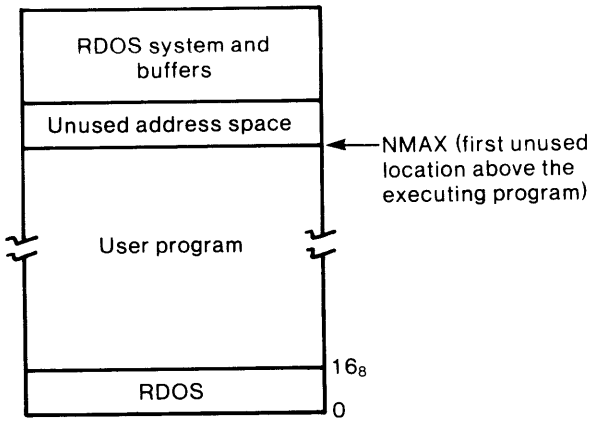


Figure 3.4 Unmapped background memory SD-00432A

The highest memory address available (HMA) is usually the first word below an unmapped RDOS system. However, the RLDR symbol table also occupies upper memory if the global switch /S was included in the RLDR command. In this case, the HMA falls directly below the symbol table; otherwise, RLDR loads its table just above your program by default.

This section discusses the .MEM and .MEMI commands, which allow you to monitor and control the amount of memory available to your programs.

.MEM

Determine available memory

This command returns the current value of NMAX in AC1, and the value of HMA in AC0. It can be followed with a SUB 1,0 instruction to determine the amount of additional memory available to your program.

In unmapped systems, HMA represents the location immediately below the bottom of RDOS—or the bottom of the symbol table, if the program was loaded or bound with the global /S switch. In mapped systems, HMA is the highest logical address available in the current program space.

Format

```
.SYSTM
.MEM
error return
normal return
```

Possible Errors

None.

.MEMI

Change NMAX

This system call allows a program to increase or decrease the value of NMAX. It updates the value of NMAX in the UST (in USTNM), and returns the new value of NMAX in AC1. RDOS does not permit an adjustment to NMAX that would cause its value to exceed HMA + 1. Nor does the system check NMAX against the original value that RLDR determined for it.

A program that requires memory space above its current NMAX can invoke the .MEMI command to allocate the number of words needed. RDOS uses the value of NMAX to determine the amount of memory to save if it suspends a program. Generally, NMAX should be updated even for temporary storage that exceeds its current value. Otherwise, the stored program may be suspended without enough information to continue. For the largest possible save file, NMAX must be a value less than or equal to 77416. As a general rule, each program should request only the amount of memory that it requires, and should release memory space when needed.

Required Input

AC0 - The incremet (positive) or decrement (in two's complement) of NMAX.

Format

```
.SYSTM
.MEMI
error return
normal return
```

Possible Errors

AC2	Mnemonic	Meaning
26	ERMEM	Attempt to allocate more memory than available.
74	ERMPR	Address outside address space.

Device Access Commands

This section describes the .DEBL and .DDIS commands, which enable or disable device access at the machine level, and the .RDSW command, which permits a program to read the position of the front panel switches or the contents of the switch register.

In mapped RDOS systems, the map unit traps if a user program attempts to access system devices such as the CPU or floating point unit (FPU). System call .DEBL makes it possible for a program to access a system device. It should be used carefully, however, since it circumvents the map unit's safeguards. Instructions such as INTDS or IORST, for example, can deactivate the system if access to the CPU is enabled.

The .DEBL command must be used in any system with floating point hardware and programs, running in two grounds, that use floating point arithmetic. Each program in such a system must enable access to the FPU so that the system can save and restore it.

In mapped NOVA systems, the .DEBL command can be issued from either ground to device code 75 or 76. The call enables access to all three FPU devices (codes 74, 75, 76). It should not be issued to device code 74 in a NOVA system.

In ECLIPSE systems, programs enable access to the FPU by issuing the .DEBL command to device code 74, unless a device such as the I/O bus is already wired to device codes 74, 75, or 76. If yours is an ECLIPSE system in which only one ground uses floating point arithmetic and a device is wired to codes 74, 75, or 76, programs can access these devices via system call .IDEF as explained in Chapter 7.

Similarly, if your system has an optional interger MPY/DVD and both programs need to use it, they must enable access via the .DEBL command, and then save and restore the MPY/DVD. In an unmapped system whose grounds will not access the FPU, the device access calls .DEBL and .DDIS are no-ops, and take the normal return. In any system, it is recommended that the .DEBL command be issued for the FPU before using it.

.DEBL

Enable user access of a device

This system call permits a program to reference any device on a machine level; it bypasses the normal system safeguards, in order to do so, and should be used carefully for that reason. The command is a no-op in unmapped systems except for hardware FPUs, as noted earlier.

Required Input

AC0 - Device code of the device to be accessed.

Format

```
.SYSTM
.DEBL
error return
normal return
```

Possible Errors

Only one possible error results from this command. Its mnemonic is ERDNM, meaning that the device code in AC0 exceeds 77 octal. The system passes error code 36 in AC2 as a result.

.DDIS

Disable user access of a device

This system call, the complement of the .DEBL command, prevents further machine-level access of a device in the system. Thus it restores the system safeguards removed if a .DEBL command was issued previously. The command is a no-op in unmapped systems, except as noted earlier.

Required Input

AC0 - Device code of the device to which user access will be disabled.

Format

```
.SYSTM
.DDIS
error return
normal return
```

Possible Errors

Only one possible error results from this command. Its mnemonic is ERDNM, meaning that the device code exceeds 77 octal, and RDOS passes error code 36 in AC2 when it occurs.

.RDSW

Read the front panel switches or register

This system call allows a program to read the position of the front panel switches or the contents of the switch register. RDOS returns the switch configuration in AC0, where bit 0 equals switch 0, bit 1 equals switch 1, and so forth. To locate the contents of the switch register in a computer with a virtual console, consult the Internal Calls table in the *Programmer's Reference* guide for your CPU.

Format

```
.SYSTM
.RDSW
error return
normal return
```

Possible Errors

None.

Clock and Calendar Commands

RDOS provides four commands to keep track of the time of day and the current date. It stores dates as days from December 31, 1967, where day one is January 1,1968. The 24-hour clock can be set by passing binary hours, minutes, and seconds in three accumulators

In order of discussion, the clock and calendar commands include:

.GTOD	Get the current time.
.STOD	Set the time of day.
.GDAY	Get the current date.
.SDAY	Set today's date.

.GTOD

Get the time of day

This command causes RDOS to pass the current time in binary form. The system returns seconds in AC0, minutes in AC1, and hours in AC2 according to 24-hour format.

Format

```
.SYSTM
.GTOD
error return
normal return
```

Possible Errors

None.

.STOD

Set the time of day

This command sets the system clock to a specific hour, minute, and second when the user passes seconds in AC0; minutes in AC1; and hours, according to 24-hour format, in AC2. All values passed must be in binary form.

Format

```
.SYSTM
.STOD
erorr return
normal return
```

Possible Errors

Only one possible error results from this command. Its mnemonic is ERTIM, signifying an illegal time of day, and the system passes error code 41 in AC2 when it occurs.

.GDAY

Get today's date

This command causes the system to return the number of the current month, day and year. RDOS returns the month in AC1, the day in AC0, and the current year—less 1968—in AC2.

Format

```
.SYSTM
.GDAY
error return
normal return
```

Possible Errors

None.

.SDAY

Set today's date

This command sets the system calendar to a specific date. The system increments the date when the time of day passes 23 hours, 59 minutes, and 59 seconds. This routine applies to the years 1968 to 2099.

Required Input

AC0 - Number of the day within the month.

AC1 - Number of the month where January is month one.

AC2 - Number of the current year less 1968.

Format

```
.SYSTM
.SDAY
error return
normal return
```

Possible Errors

Only one possible error results from this command. Its mnemonic is ERTIM, signifying an illegal day, month or year, and RDOS returns error code 41 in AC2 when it occurs.

Spooling Commands

SPOOL is an acronym for simultaneous peripheral operation on line. RDOS automatically spools data output to devices $DP0, $LPT, $LPT1,$PTP, $PTP1, $TTO, $TTO1, $TTP, and $TTP1. Spooling to plotter devices $PLT and $PLT1 must be explicitly enabled.

The system performs spooling by queuing data on disk for one or more spoolable devices to receive, leaving the CPU available for further processing. This procedure occurs only when no other system operations are ready, and is controlled by means of the system calls described in this section. In order of discussion, they are the .SPKL, .SPDA, and .SPEA commands.

Spooling requires that you include, during system generation, at least two stacks for a single-program environment, and at least three system stacks for a dual-program environment. All spooling commands become inoperative if the number of stacks specified is insufficient for RDOS to execute them. Spooling also requires disk buffering, an operation for which RDOS dynamically allocates space from the master directory. The system temporarily disables spooling if it requires more disk space for its buffers than is currently available. Spooling operations can be re-enabled when more disk space is free.

.SPKL

Stop a spool operation

This command halts a current spool operation for a given device. All data on the output queue is forfeited as a result.

Required Input

AC0 - Byte pointer to the name of the device receiving spooled data.

Format

```
.SYSTM
.SPKL
error return
normal return
```

Possible Errors

AC2	Mnemonic	Meaning
1	ERFNM	Illegal filename.
3	ERICD	Illegal command for device.
36	ERDNM	Device not in system.
74	ERMPR	Address outside address space.

.SPDA

Disable Device Spooling

This command stops a device from spooling its output. If issued while a device is spooling, execution is delayed until RDOS has completed the spooling operation. Data output to the device before the spooled data has been exhausted will itself be spooled, delaying execution of the .SPDA command even longer.

Required Input

AC0 - Byte pointer to the device for which spooling will be disabled.

Format

```
.SYSTM
.SPDA
error return
normal return
```

Possible Errors

AC2	Mnemonic	Meaning
1	ERFNM	Illegal filename.
3	ERICD	Illegal command for device.
36	ERDNM	Device not in system.
74	ERMPR	Address outside address space.

.SPEA

Enable device spooling

This system call enables spooling after it has previously been disabled for a given device. RDOS itself may have disabled spooling because of insufficient disk space, or a user may have stopped spooled operations with system call .SPDA or CLI comand SPDIS.

Required Input

AC0 - Byte pointer to the device name.

Format

```
.SYSTM
.SPEA
error return
normal return
```

Possible Errors

AC2	Mnemonic	Meaning
1	ERFNM	Illegal filename.
3	ERICD	Illegal command for device.
36	ERDNM	Device not in system.
74	ERMPR	Address outside address space.

Keyboard Interrupts

Programs that run under RDOS can be interrupted by typing certain control characters from the console. This book represents control characters as CTRL-*x*, where *x* is an alphabetic character that is pressed simultaneously with the CTRL pad on your keyboard. Typing CTRL-A or CTRL-C interrupts a background program from the console, while typing CTRL-F halts a foreground program from the background console.

The control characters CTRL-A and CTRL-F work abruptly: they halt program execution in their respective grounds, save nothing, and pass control to the higher-level program—generally the CLI. The control character CTRL-C writes the current core image to disk file BREAK.SV—or to FBREAK.SV, if you issued CTRL-C from the foreground console—and passes control to the CLI. The system returns the message -INT to your console after executing CTRL-A, or the message BREAK after executing CTRL-C. Chapter 6 explains the effects of CTRL-F in more detail.

Interrupts can also be programmed with the aid of the system calls described in this section. In order of discussion, they are:

.BREAK	Interrupt a program and save the state of main memory.
.ODIS	Disable console interrupts.
.OEBL	Enable console interrupts.
.INTAD	Assign a task to service keyboard interrupts.

To pass control to a program other than the CLI after a keyboard interrupt, you must set up its user status table, or UST, as described next. Note that processing of the .BREAK command is invalidated after an exceptional status condition.

Defining Interrupt Routines

For each program level, the system creates a *user status table* (UST). Each UST is 24_8 words long and resides in user address space, starting at location 400 octal. A UST includes two words, USTIT and USTBR, which contain addresses for interrupt routines that service CTRL-A and CTRL-C. Word USTIT contains the address of the routine that gains control after you enter CTRL-A, while word USTBR holds the address of the routine for CTRL-C. Both words are initialized to —1 when you load a program. This number must be changed in order to specify your own interrupt routines. Note, however, that the value of word USTBR is set to zero whenever RDOS passes control to its address as a result of a user trap. Thus, if a program is to receive control at address USTBR after traps, it must reset that address after each trap occurs. Chapter 5 describes the user status table in more detail.

If word USTIT contains —1 when you hit CTRL-A, or if word USTBR holds —1 when you type CTRL-C or issue the .BREAK command, the system closes all channels on the current level and loads the next higher level program. Then RDOS checks this level's UST for the address of an interrupt routine; it will not pass control to the address of a user-defined break routine if that adddress is less than 16 octal. The system continues this process until it reaches a program and level whose UST contains the address of an interrupt routine. If it reaches the CLI on level 0, it uses the CLI's routine. If you have chained from the CLI, however, and the new program at level 0 contains no address for an interrupt routine, the system halts in an exceptional status condition as explained in Appendix E.

During its search for the address of an interrupt routine, the system checks each program level for a TCB queue. If the queue is missing—perhaps because you accidentally overwrote it or because it is in user address space—the system skips this program and examines one at the next-higher level.

After finding a program with words USTIT or USTBR set to an address other than -1, RDOS checks word USTIA, also contained in the UST, to find a task's TCB address. Although the loader initializes word USTIA zero, it may contain a TCB address under certain conditions.

If word USTIA contains zero, the system appropriates the TCB of the task (pointed to by USTAC) whose priority is currently highest; transfers that task's PC to temporary storage (TTMP in the TCB); and places the UST's interrupt address in TPC. (TPC is the program storage counter in the TCB.) Control then passes to the scheduler, which launches the task of highest priority. Since the UST's interrupt address is placed in TPC of the highest priority task, RDOS executes the interrupt routine. (A single-task program is itself the highest priority task.) Figure 3.5 shows a program with an interrupt handler.

```
START:  LDA 2, USTP       ; Put UST address in AC2.
        LDA 0, .BRKA      ; Pointer to address of
                          ; CTRL-A handler.
        STA 0, USTIT, 2   ; Store CTRL-A address
                          ; in USTIT.

; The main program follows here.

MAIN:   ....
        ....
        ....

; CTRL-A handler, whose code will be
, executed on CTRL-A.

BRKA:   ....
        ....
        ....

.BRKA:  BRKA
        .
```

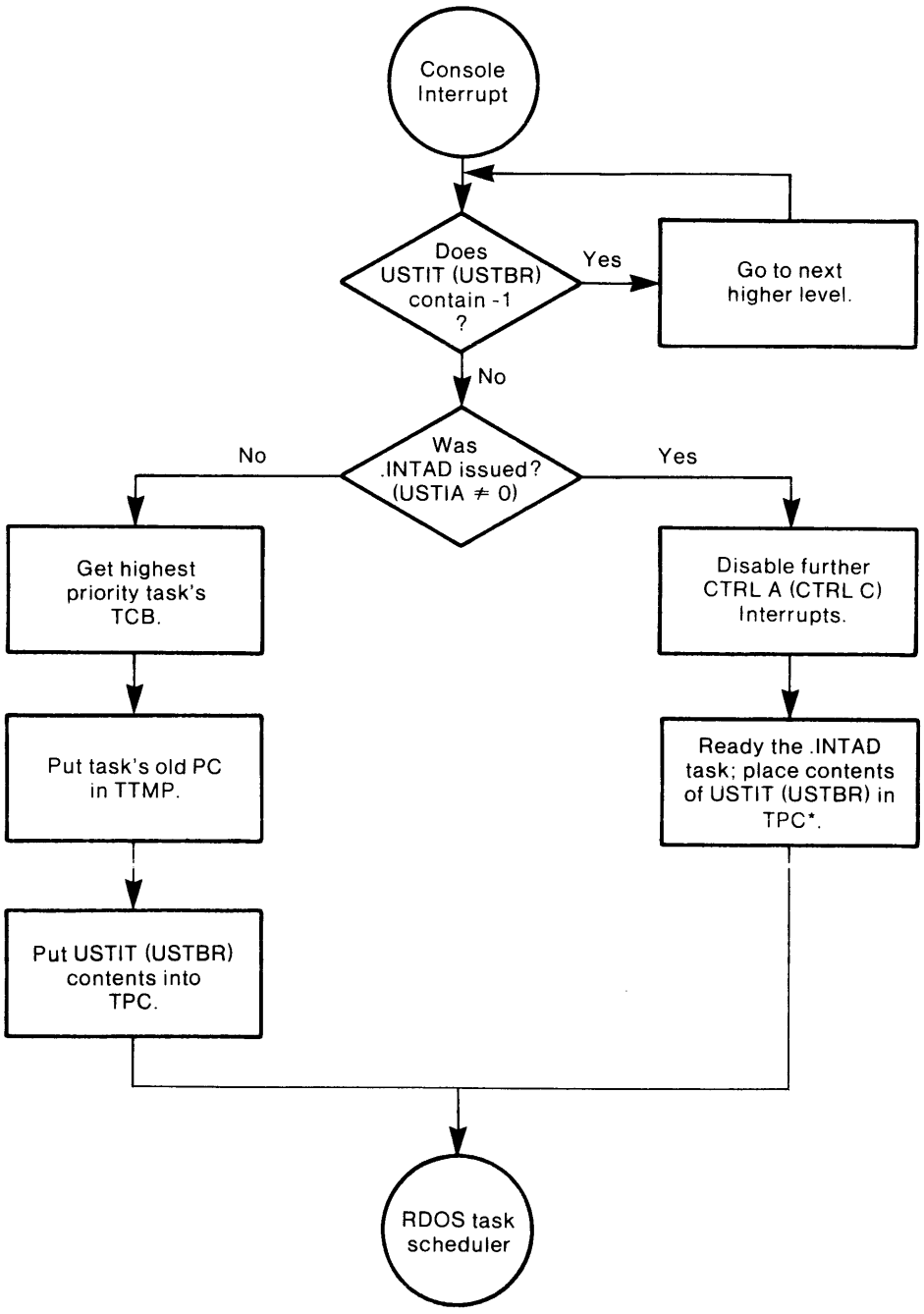
Figure 3.5 Program with interrupt handler

If a task issues system call .INTAD before the interrupt, RDOS finds a nonzero value in word USTIA. This value is the issuing task's TCB address. RDOS then readies the issuing task and stores the value of USTIT or USTBR in its TPC. Next, the system disables further interrupts via CTRL-A or CTRL-C, and passes control to the Task Scheduler. When the .INTAD task gains control, it executes the appropriate interrupt service and reenables console interrupts—if desired—by issuing system calls .INTAD or .OEBL, described later in this section. Note that your main program should not issue the .INTAD command unless you want it to suspend itself. Figure 3.6 shows a program containing an .INTAD task, while Figure 3.7 shows the logic of program interrupts in flow chart form.

The break file created by an interrupt via CTRL-C or the .BREAK command is a save file. This file contains the current state of main memory, from SCSTR (the start of save files, location 16) through the highest of NMAX or the start of the symbol table, SST. RDOS creates the break file in the current directory under the filename BREAK.SV, or under FBREAK.SV if the foreground program issued a .BREAK command. The system deletes any existing break file before writing a new one. If RDOS cannot write a break file, possibly because it lacks sufficient file space on disk, control passes to the address specified in USTBR *less one location*, and the system returns an error code in AC2. If disk space is insufficient for a new break file, RDOS uses the available disk blocks but lists this file as zero bytes. To release these blocks, delete the file.

```
; The main task creates the .INTAD task and
; initializes the CTRL-A processing address.

START:  SUB 0,0          ;0 priority for .INTAD task.
        LDA 1, .INTSK    ; Add of .INTAD task.
        .TASK            ; Create the .INTAD task.
        JMP ER           ; Mandatory.
        LDA 2, USTP      ; Put UST address in AC2.
        LDA 0, .ROUT1    ; Name of CTRL-A routine.
        STA 0, USTIT,2   ; Put CTRL-A routine in address
                         ; of .INTAD task in USTIT.

; The main program follows here.

MAIN:   ......
        ......
        ......

.INTSK: INTSK
        ....

.ROUT1: ROUT1

; This is the INTAD task.
INTSK:  .SYSTM           ; On program execution, the .INTAD
        .INTAD           ; task issues .INTAD, suspending
                         ; itself until CTRL-A is entered.
        JMP ER           ; System never takes error or
        JMP INTSK        ; normal return from .INTAD
.ROUT1: ....             ; On CTRL-A, the .INTAD task
        ....             ; executes this code.
        ....
        JMP INTSK        ; After performing its routine, the
                         ; .INTAD task reissues .INTAD,
                         ; thereby suspending itself and
                         ; reenabling CTRL-A interrupts.

.ER:    .SYSTM           ; If program reads from console,
        .ERTN            ; error handler must pass EREOFs,
        JMP .            ; since CTRL-A and CTRL-C supply
                         ;EOFs to console.
        .END START
```

Figure 3.6 Program with .INTAD task



*If break fails, USTBR-1 is placed in TPC.

Figure 3.7 Program interruption logic sequence

SD-00753

Although the break file is, in essence, a snapshot of main memory's current state, the file is not directly executable; it is generally useful for debugging. Before attempting to execute it, you must consider how the interruption generated by CTRL-C or the .BREAK command has affected the system:

1. It closed all open channels, requiring that you reopen them if needed by the break file.

2. It purged all .DELAY commands, yet their tasks remain suspended.

3. It removed all user-defined clocks and interrupt devices, which must now be redefined if you require them.

4. It destroyed all read-operator messages.

5. It disabled your access to all devices enabled via the .DEBL command, including the floating point unit. Access must be reenabled if the break file needs these devices.

Keyboard interrupts are enabled by default when you execute a program. RDOS provides two system calls, .ODIS and .OEBL, to disable or reenable further keyboard interrupts. Neither call affects the .BREAK command, which performs the same operation as CTRL-C. To restore keyboard interrupts after an .INTAD operation—and any interrupt via CTRL-A, CTRL-C, or the .BREAK command—the .INTAD task must issue the .OEBL command or another call to .INTAD.

.BREAK

Interrupt program and save main memory

This system call is operationally equivalent to typing CTRL-C on the console. It saves the state of memory in save file format from location 16 to the highest of NMAX or the start of the symbol table, SST. The filename used is BREAK.SV, or FBREAK.SV if the command was issued by the foreground program). Any previous break file is deleted before the new one is written to the current directory, where you may retain it, save it under another name with the CLI's SAVE command, or delete it. Generally, because system breaks close all channels, the break file is useful only for debugging with a disk editor such as OEDIT or SEDIT.

The break file that results from an interruption via CTRL-C or BREAK saves the program in the following state:

- All open channels are closed.
- All .DELAY commands have been purged, while their tasks remain suspended.
- Blocks and interrupt devices defined by the user are removed.
- All read-operator messages are lost.
- All user accesses enabled via the .DEBL command are lost.

Unlike its console counterpart CTRL-C, the .BREAK call is operative at all times and the .ODIS command, described later, cannot disable it.

As explained earlier, if word USTBR contains a valid address, control passes to this address after RDOS writes the break file to disk. If USTBR contains —1, RDOS searches the user status tables of programs at progressively higher levels until it finds a valid address in word USTBR. Control goes to the first higher-level program whose USTBR contains such an address. If RDOS cannot write the break file due, for example, to insufficient file space, control passes to the address contained in word USTBR less one location.

Format

```
.SYSTM
.BREAK
```

There are no standard error or normal returns.

Possible Errors

AC2	Mnemonic	Meaning
27	ERSPC	Out of disk space.
60	ERFIN	BREAK.SV (or FBREAK.SV) is in use.
101	ERDTO	Disk timeout occured.

.ODIS

Disable console interrupts

This command disables console interrupts within a program. When issued from the background, it disables interrupts via CTRL-A and CTRL-C. When issued from the foreground, it disables interruptions that result from CTRL-A and CTRL-C, and CTRL-F. Operations that issue from the .BREAK command cannot be disabled with this system call. The .OEBL, presented next, reenables console interrupts when issued from your program.

Format

```
.SYSTM
.ODIS
error return
normal return
```

Possible Errors

None.

.OEBL

Enable console interrupts

When you first bootstrap a system, RDOS enables console interrupts via CTRL-A, CTRL-C, and CTRL-F. If you disable console interrupts by system call .ODIS or by processing a console interrupt with an .INTAD task, this call reenables them within its program environment.

Format

```
.SYSTM
.OEBL
error return
normal return
```

Possible Errors

None.

.INTAD

Reserve a program interrupt task

This system call enables keyboard interrupts and permits you to assign a task to service interrupts from CTRL-A, CTRL-C, and the .BREAK command. The servicing task must issue the .INTAD call; RDOS will recognize it as the interrupt task. Because the .INTAD command causes the issuing task to suspend itself, the servicing and main tasks are generally not the same; that is, the main program or task should not issue this command. RDOS uses the .INTAD task (instead of a program task's TCB) for interrupts, thereby preserving the current program environment aside from any system call executing when the interrupt occurs.

Format

```
.SYSTM
.INTAD
error return
normal return
```

Possible Errors

None.

Summary

This section summarizes all commands discussed in this chapter in Table 3.9.

System Call	Function
.APPEND	Open a file for appending.
.BREAK	Interrupt the current program and save the current state of memory in save file format.
.CCONT	Create a contiguously organized file with all data words zeroed.
.CONN	Create a contiguously organized file with no zeroing of data words.
.CDIR	Create a subdirectory.
.CHATR	Change file attributes.
.CHLAT	Change link access attributes.
.CHSTS	Get the status of the file currently open on a specified channel.
.CLOSE	Close a file.
.CPART	Create a secondary partition.
.CRAND	Create a random file.
.CREAT	Create a sequential file.
.DDIS	Disable user access to a device in a mapped system.
.DEBL	Enable user access to a (mapped) system device.
.DELAY	Delay the execution of a task.
.DELET	Delete a file.
.DIR	Change the currrent directory.
.DUCLK	Define a user clock.
.EOPEN	Open a file for reading and writing by one user only.
.EQIV	Assign a temporary name to a device.
.ERTN	On an error, return from program and describe error (if to CLI).
.GCHAR	Get character from the console.
.GCHN	Get the number of a free channel.

Table 3.9 System call summary

System Call	Function
.GCIN	Get the operator input console name.
.GCOUT	Get the operator output console name.
.GDAY	Get today's date.
.GDIR	Get the current directory name.
.GPOS	Get the current file pointer.
.GSYS	Get the name of the current operating system.
.GTATR	Get file attributes.
.GTOD	Get the time of day.
.IDEF	Identify a user device.
.INIT	Initialize a device or a directory.
.INTAD	Define a program interrupt task.
.LINK	Create a link entry.
.MDIR	Get the logical name of the master device.
.MEM	Determine available memory.
.MEMI	Change NMAX.
.MTDIO	Perform free format I/O on tape or cassette.
.MTOPD	Open a magnetic tape or cassette for free format I/O.
.ODIS	Disable keyboard interrupts for this console.
.OEBL	Enable keyboard interrupts for this console.
.OPEN	Open a file for reading and/or writing by one or more users.
.OVLOD	Load a user overlay into memory.
.OVOPN	Open a user overlay file.
.OVRP	Replace an overlay file.
.PCHAR	Write a character to the console.
.RDB	Read one or more disk blocks.
.RDL	Read a line.
.RDR	Read a random record.
.RDS	Read sequential bytes.

Table 3.9 System call summary (continued)

System Call	Function
.RDSW	Read the console switches.
.RENAM	Rename a file.
.RESET	Close all files.
.RLSE	Release a directory or device.
.ROPEN	Open a file for reading only by one or more users.
.RSTAT	Get a resolution file's statistics.
.SDAY	Set today's date.
.SPDA	Disable spooling.
.SPEA	Enable spooling.
.SPKL	Delete the current spool file.
.SPOS	Set the current file pointer.
.STAT	Get a file's statistics.
.STOD	Set the time of day.
.ULNK	Delete a link entry.
.UPDAT	Update the current file size.
.VMEM	Determine the number of memory blocks.
.WRB	Write one or more 256-word blocks to disk.
.WRL	Write a line.
.WROPR	Write an operator message.
.WRR	Write a random record.
.WRS	Write sequential bytes.

Table 3.9 System call summary (continued)

Chapter 4

Extending User Address Space

Occasionally a program will require more memory than is available in the computer. This chapter introduces the facilities that RDOS provides for augmenting the limits of main memory, and explains how to use them. Its two major sections cover the following subjects:

- Program swapping and chaining
- User overlays
- Memory protection
- Virtual user overlays
- Window mapping
- Extended direct block I/O

All system calls described in these sections are summarized at the end of the chapter in table form.

Program swaps, chains, and user overlays are tools that effectively extend main memory with *disk space*. These tools apply to all systems and applications, and must be understood in order to write advanced programs in RDOS.

When a program swaps or chains, it calls another program into execution. During this process, the same areas of your address space can be used for diverse operations.

Program swapping is executed from one of four RDOS levels of control, where one level calls another.

Chained programs are called in sequence by a program on the same level, and overwrite the calling program.

Overlays also operate on one level, but are called in succession by a core-resident root program and placed in a reserved area (node) of memory.

In any Data General computer, mapped or unmapped, the directly addressable memory available to a program cannot exceed 32K words. Naturally, this depends on the total amount available in the machine. In a dual-program environment, each of two programs may use up to 32K of this space, known as *logical address space* in mapped systems.

Mapped RDOS permits a program in either a single or dual environment to access memory outside its logical address space. This supplementary area of memory is called *extended address space*, or *extended memory*. The total address space (both logical and extended) is allotted to a program in a mapped system via the CLI's SMEM command.

Mapped RDOS offers two programming tools for manipulating extended address space: window mapping and virtual overlays. *Window mapping* is most useful for extended data storage, made possible by a window map defined by your program. It also allows you to transfer 256-word blocks of data via extended direct block I/O. *Virtual overlays*, like conventional ones, are most useful for storing subroutines, and are defined via utility RLDR. Both features can be implemented in one program.

The tools of memory protection and extended direct block I/O are also available for programming with extended memory, and the sections that discuss them apply to users of mapped RDOS only.

Program Swapping and Chaining

This section discusses the operations of swapping and chaining, along with the system calls that enable you to implement these operations in your programs. In order of discussion, the system calls are:

.EXEC	Swap or chain a save file into execution.
.RTN	Return to the next higher level program.
.ERTN	Return from a program swap with the error status of the calling program.

Any program executing under RDOS can suspend its own execution and swap in another program, or chain to an executable segment of itself. Occasionally this book uses the term *push* instead of *swap*. The terms are synonymous, meaning to execute a program on the next lower level via the .EXEC command. The CLI command POP, which instructs RDOS to execute the program on the next higher level, corresponds roughly to system call .RTN.

Programs with open multiplexor lines must close them before swapping; otherwise, they will take the error return from system call .EXEC. Note that any program you plan to swap or chain must be an executable save file.

Program swaps may exist in up to five levels, where one level calls for another and the Command Line Interpreter exists at the highest level, level 0. The CLI is merely one program that RDOS can execute. Its only special property is that it normally executes at the highest level in the system. Generally, the utilities supported by the CLI—that is, the text editors, assemblers, and binder or loader—execute at level 1. When you execute a program or utility from the CLI, RDOS commonly swaps the CLI to disk and calls it back automatically, via the .RTN command, after the program has completed its execution. Figure 4.1 illustrates the swapping process.

Alternatively, a large program can be composed of a sequence of executable segments in which the end of each segment invokes the beginning of the next, ending with the CLI. This process, called chaining, occurs on one level. The length of the entire program is limited only by the disk space available to it. A program chain can be invoked with system call .EXEC or, from the console, via the CLI's CHAIN command. Figure 4.2 diagrams the chaining process.

When a program issues the .EXEC command, a swap or chain occurs depending on your input in AC1. If a swap is specified, RDOS saves a core image of the current program; brings the new program, specified in AC0, into main memory; and executes this program. The calling program's task control block (TCB) saves its accumulators, carry, and PC. The new program can swap itself and execute the original one by issuing the .RTN or .ERTN commands, or it may swap to a lower level by issuing system call .EXEC. Any program can check its current level via the .FGND command discussed in Chapter 4.

If AC1 specifies a chain, RDOS brings the program specified in AC0 into core and executes it. The system does not save a core image of this program. After it has finished, this program can launch any other into execution via the .EXEC command.

When planning program swaps, make sure that NMAX accurately reflects core memory for every program in use. Remember that during a swap RDOS saves the current core image up to the higher of NMAX or SST (start of the user symbol table). Thus, if your program exceeds NMAX and invokes another program, RDOS can save only a portion of the calling program's memory state; the remainder of the calling program will be lost. Even if the executing program does not call another, a break from your console may force suspension. To avoid these problems, your program should never use temporary storage at load time above its original value of NMAX without first instructing the system to allocate more memory for this purpose. The .MEMI command, discussed in Chapter 3, performs this function.

The operations of swapping, chaining, and returning halt activity in the current program. RDOS terminates calls and conditions that would not be appropriate in the new program, most of them involving multitask activity. The following conditions are terminated when a change of program occurs; to restore them, refer to the appropriate chapter and system call, as indicated:

1. A return or chain closes all channels, requiring the new program to open the channels it needs as described under Input/Output Commands in Chapter 3. When the calling program's execution resumes after a swap, all channels that were open when the swap occurred will be open.

2. All $TTI or $TTI1 input is halted. The system calls affecting this condition include .GCHAR, discussed in Chapter 3; .TRDOP, discussed in Chapter 5; and .RDOP, discussed in Chapter 6.

3. Any system devices enabled for user access via the .DEBL command (Chapter 3) are disabled. Thus, the new program must enable access to the hardware floating point unit, if one is present.

4. Console interrupts are enabled, cancelling any outstanding instructions to disable them via the .ODIS command (Chapter 3).

5. The state of the floating point unit is not preserved.

6. All interrupt message transmissions are removed. Refer to the .IXMT command in Chapter 5 for details.

7. If you have defined a user clock or a system delay, it is removed. Consult the .DUCLK and .DELAY commands in Chapter 5 for details.

8. If your system has operator messages, the state of the OPCOM, discussed in Chapter 5, is lost.

9. All user-defined interrupt service is removed, as is any mapped system data channel map setting in a mapped system. See the discussions of .IDEF and .STMAP in Chapter 7 for details.

10. Mapped systems only: all write-protection of mapped memory, defined via the .WRPR command (Chapter 4), is removed.

11. Mapped systems only: extended space reserved for virtual overlays is released, and all definitions of extended memory made via window mapping are removed.

12. Mapped systems only: any dual-program communications area, defined via system call .ICMN (Chapter 6), is removed.

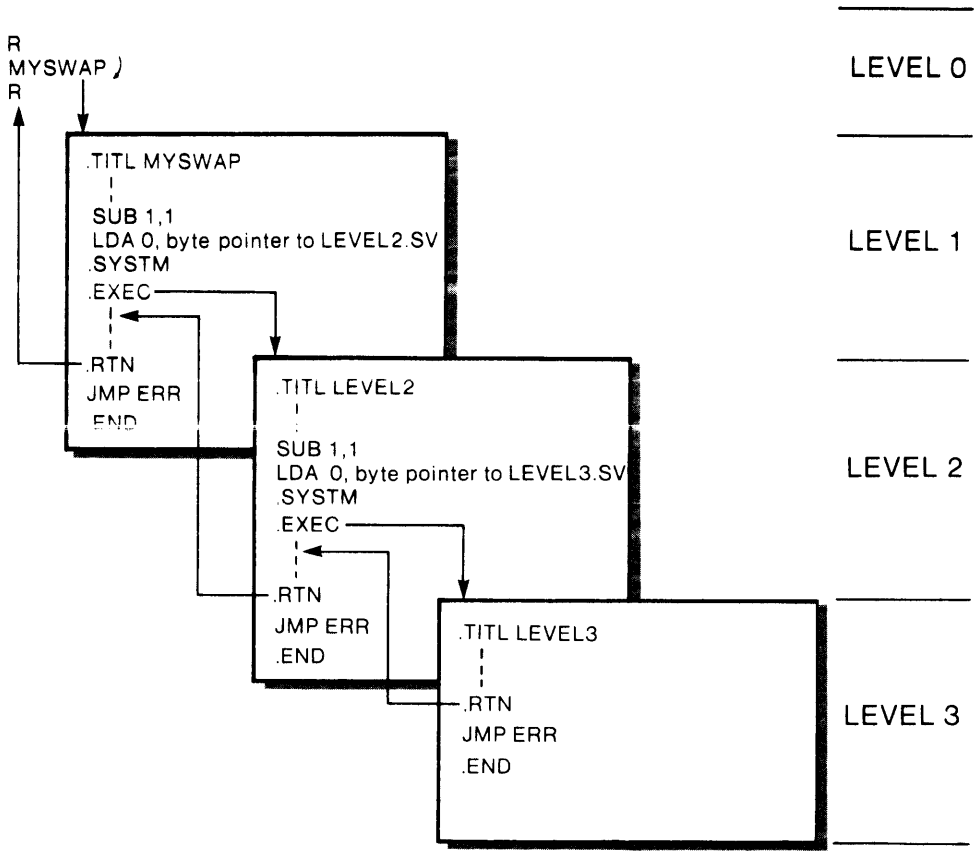


Figure 4.1 Program swapping

SD-00504

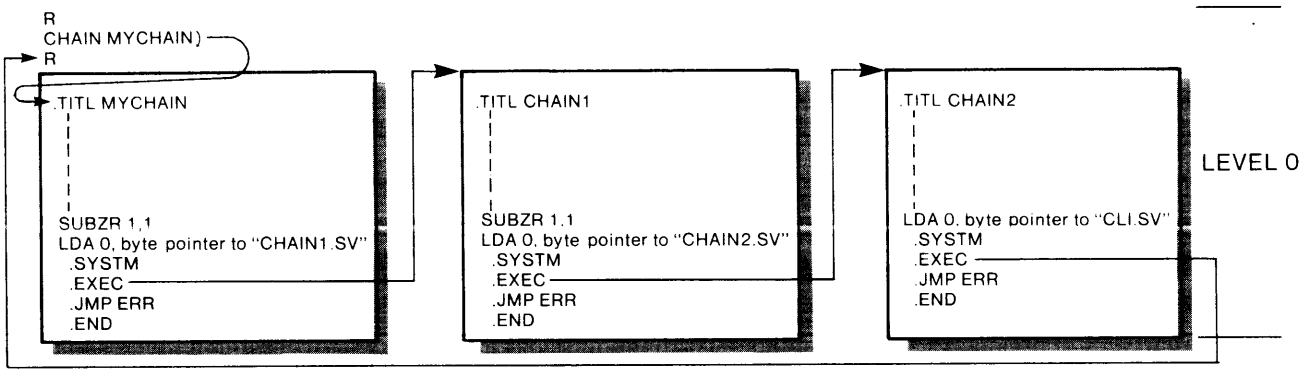


Figure 4.2 Program chaining

SD-00505

.EXEC

Swap or chain a save file into execution

This command requests the system to swap or chain a program. See Figures 4.1 and 4.2, shown earlier, illustrate each process.

Required Input

AC0 - Byte pointer to filename of new program (save file).

AC1 - Specifies a code for swap or chain, as shown in the following table.

Code in AC1	Meaning
0	Swap to user program. Control goes to the ready task with highest priority.
1B0	Chain to user program.
1	Swap and start at debugger address.
1B0 + 1B15	Chain and start at debugger address.

The code in AC1 indicates one of two starting addresses: the program starting address (USTSA), and the Debug III starting address (USTDA). Chapter 5 discusses these addresses in detail.

Note that if bit 0 of AC1 is 1, RDOS does not save the current level, and the operating level remains unchanged. This feature provides unlimited program chaining. Also note that you cannot swap from the foreground of an unmapped system. An attempt to do so causes RDOS to return error code 25 (ERCM3). You can, however, chain from an unmapped foreground provided the new program's memory requirement is less than or equal to that of the previous program.

The new program receives the contents of AC2. If this program is the CLI (CLI.SV) and AC2 contains a nonzero value, the CLI searches its special command file, CLI.CM, for commands. This mechanism is fully described in *RDOS/DOS Command Line Interpreter*.

Format

```
.SYSTM
.EXEC
error return
normal return
```

Possible Errors

AC2	Mnemonic	Meaning
1	ERFNM	Illegal filename.
4	ERSV1	File requires save attribute (S).
12	ERDLE	File does not exist.
25	ERCM3	More than five swap levels, or swapping from unmapped foreground.
26	ERMEM	Attempt to allocate more memory than is available.
32	ERADR	Illegal starting address.*
53	ERDSN	Directory specifier unknown.
57	ERLDE	Link depth exceeded.
66	ERDNI	Directory not initialized.
73	ERUSZ	Too few channels defined at load time or during system generation.
74	ERMPR	Address outside address space.
101	ERDTO	Disk timeout occurred.
102	ERENA	No linking allowed (N attribute).
125	ERNSE	Program not swappable.

*RDOS returns ERADR status if (1) no starting address was specified for the save file and bit 15 is reset to 0, or (2) the debugger was not loaded as part of the save file and bit 15 is set to 1.

.RTN

Return to the program at the next higher level

This system call closes all open channels and returns to the calling program at its normal return point. All the calling program's accumulators are restored, and control passes to the instruction at the return point. If the level 0 foreground program issues this command, RDOS closes all foreground channels, releases the foreground, and displays the message FG TERM on the background console.

Format

```
.SYSTM
.RTN
error return
```

Normal returns are precluded by the fact that RDOS restores the calling program in memory. The error return, reserved for compatibility with RTOS, is never taken. Error conditions cause exceptional system status, as explained in Appendix E.

.ERTN

Return from program swap with calling program's error status

This command instructs a called program to return error information to its caller, enabling you to determine why a swapped program took the error return. The command is identical to system call .RTN, except that normal return is made to the error return of the higher-level program. Upon return, AC2 contains the value for the lower-level program instead of the value for the higher-level program. A single word of status can therefore be returned.

If a program issuing the .ERTN command executes at level 1 and returns to the CLI, the CLI outputs an appropriate message concerning the status code in AC2. The CLI prints a textual message if it recognizes a system error code; the error ERDLE, for example, corresponds to system error code 12 and evokes the message FILE DOES NOT EXIST. If RDOS returns null error ERNUL (code 20) in AC2, the CLI reports no error message.

Note that if the called program passes error EREXQ (code 17) in AC2, the CLI takes its next command from disk file CLI.CM. If the CLI does not recognize the code, RDOS displays the message UNKNOWN ERROR CODE *n*, where *n* is the numeric code in octal.

Format

```
.SYSTM
.ERTN
error return
```

The error return, reserved for compatibility with RTOS, is never taken. Error conditions cause exceptional system status, as explained in Appendix E.

User Overlays

This section explains how to extend your memory resources with user overlays, which apply to mapped and unmapped systems alike. After a thorough examination of how user overlays are constructed, the system calls that control them are presented. In order of discussion, these calls are:

.OVOPN	Open an overlay file on a specified channel.
.OVLOD	Load an overlay into the area of memory reserved for it.
.OVRP	Replace the overlays in an overlay file.

User overlays are blocks of code, placed in an overlay file, that support a root program. The root program is a save file that remains in memory throughout a program level; it extends from location 16_8 to NMAX, and calls overlays from disk into core memory as required. The overlay file is contiguously organized, and divided into segments. Each segment contains the overlays that the root program will load, one at a time, into a reserved area of memory called a *node*.

The RLDR command loads the root program; creates the overlay file; places overlays into segments of the file; and sizes the reserved area of memory, or node. If you specify overlays in the RLDR command line, the loader program produces a save file, *filename*.SV, and an overlay file, *filename*.OL, where *filename* is the name of the first binary in the command line unless you specify otherwise with switches.

To use overlays, your program must (1) open the overlay file on an RDOS channel vis system call .OVOPN, and (2) instruct RDOS, via system call OVLOD to load one overlay at a time from a segment into its node. The node is reserved for the overlays in its segment until the program terminates. Your program can free the channel by closing it via the .CLOSE command. (This process differs slightly for a multitask program, as explained under "User Overlay Management" in Chapter 5.) Appendix C demonstrates the use of overlays in a real-time programming example that includes a root program supporting two overlays.

The size of each node is the smallest multiple of 400_8 words large enough to contain the largest overlay in the node's segment. Any overlay that is not the same size as its node will be padded out with zeroes. This means that any segment size equals the node size multiplied by the number of overlays within the segment. Each segment is identified on disk by its corresponding node number.

An overlay file can hold up to 124 overlay segments; a segment may contain a maximum of 125 overlays; and each overlay can be as large as 126 disk blocks, or 31,256 words, in size. When a segment contains overlays dissimilar in length, considerable disk space will be used to pad out the smaller overlays to standard size; likewise, valuable memory is consumed to pad out the core node. for this reason, you should place overlays of roughly equal size in the same segment whenever possible.

Directory information for each overlay resides in an overlay directory. RLDR builds this directory into your program's save file, as explained in Appendix D. Each overlay has a label which the system uses to identify it; the label resolves to a *node number* and an *overlay number*, packed by halfwords.

The format used to create an overlay file and associate it with a root program is explained under the RLDR command in *RDOS/DOS Command Line Interpreter*. The following command line, examined in conjunction with Figure 4.3, serves as an example:

```
RLDR RO [A,B,C,D] R1 R2 [E,FG,H] ⟨CR⟩
```

As Figure 4.3 shows, this statement creates a disk save file, R0.SV, and an overlay file, R0.OL. The save file contains R0, R1, and R2, along with vacant areas, or nodes, for the overlays in each segment. The overlay file contains seven overlays—binary versions of A, B, C, D, E, F, G, and H. These overlays are grouped in two segments of overlay file R0.OL, where Segment 0 contains overlays A through D (numbered 0 through 3)—all destined for node 0 in main memory—and Segment 1 contains overlays E through H (numbered 0 through 2)—which will occupy node 1 in core. Note that the order in which you specify the overlay binaries in the command line determines both the overlay number and node number of each overlay.

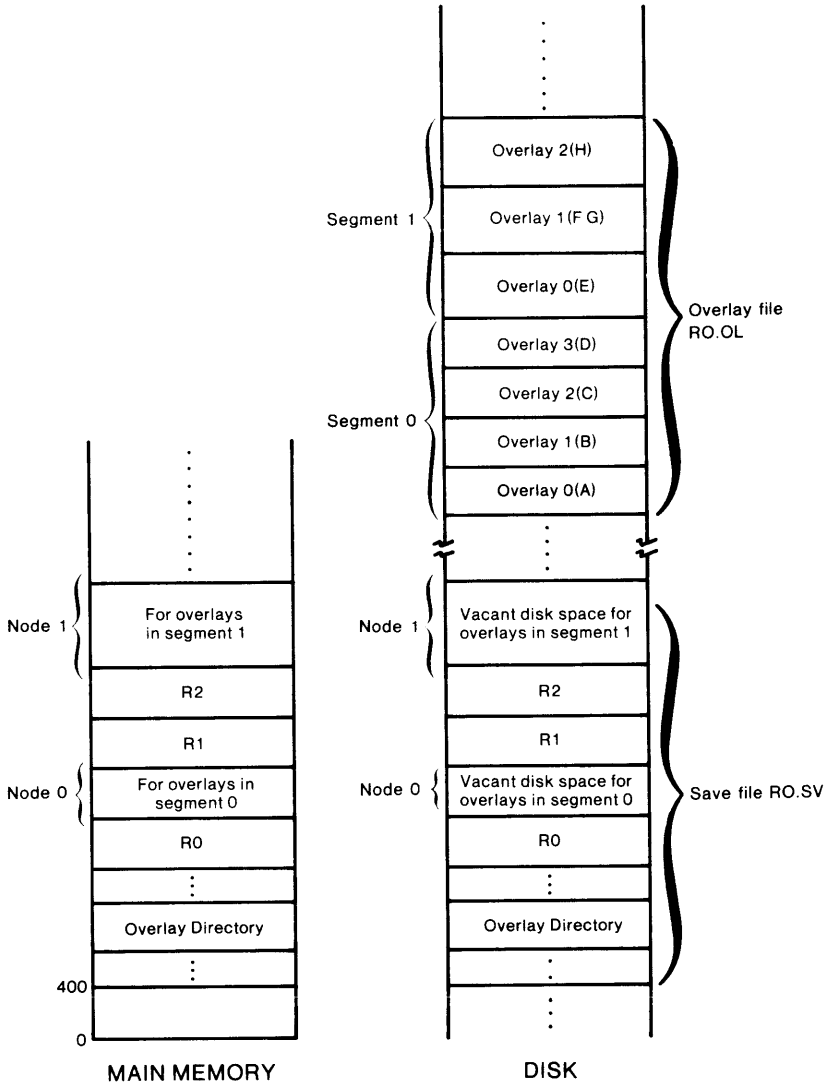


Figure 4.3 User overlays

SD-00498

You may disregard the loading order of overlays in a node if you use pseudo-op .ENTO. (Appendix C demonstrates the use of .ENTO in a real-time programming example.)

This pseudo-op allows you to assign a unique name to each overlay, rendering the order of overlays in the RLDR command line unimportant. Each binary is given a unique name in argument to .ENTO, and is referenced by that name in your program. All unique labels must be declared with pseudo-op .EXTN. Whenever .ENTO has not been used, your RLDR command line must list overlay binaries in their proper order.

Figure 4.4 takes a closer look at the overlay file, RO.OL, that resulted from our sample RLDR command line. It focuses on Segment 1 to show some of the possible entry points in overlays E, F, G, and H. Note that these binaries have been assigned unique names via pseudo-op .ENTO, eliminating the need to know which node and overlay number corresponds to them. Thus, our sample RLDR command line could have read:

```
RLDR RO [A,B,C,D] R1 R2 [H,E,F G] ⟨CR⟩
```

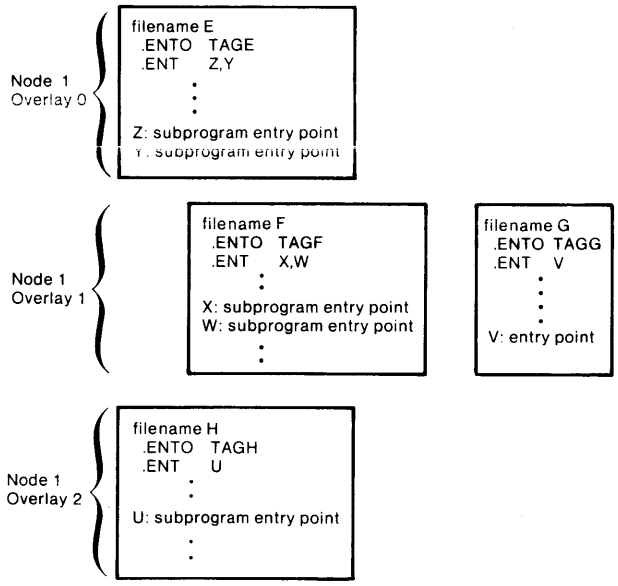


Figure 4.4 Segment 1 of overlay file R0.OL SD-00533

Figure 4.5 shows how save file RO.SV uses unique labels, created via .ENTO, to load each overlay root program. When RO.SV issues system call .OVLOD, RDOS loads all of binary E into core, where routines Z and Y serve as entry points.

```
.EXTN  TAGE, TAGF, TAGG, TAGH
.EXTN Z, Y, X, W, V, U

.OVE:   TAGE        ;TAGE IS RESOLVED TO
                    ;NODE 1, OVERLAY 0
                    ;(ENCODED AS 400).
.OVF:   TAGF        ;TAGF IS RESOLVED TO
                    ;NODE 1, OVERLAY 1
                    ;(ENCODED AS 401).
.OVG:   TAGG        ;TAGG IS RESOLVED TO
                    ;NODE 1 OVERLAY 1
                    ;(ENCODED AS 401).
.OVH:   TAGH        ;TAGH IS RESOLVED TO
        .           ;NODE 1, OVERLAY 2
        .           ;(ENCODED AS 402).

        LDA 0,bptr-to-R0.OL
        .SYSTM
        ;OPEN R0.OL ON
        .OVOPN 3        ;CHANNEL 3.
        JMP ERR         ;ERROR RETURN.
        LDA 0, .OVE     ;GET OVERLAY NUMBER.
        ADC 1,1         ;PREPARE FOR UNCON-
                        ;DITIONAL LOAD.
        .SYSTM          ;LOAD OVERLAY E
        .OVLOD 3        ;UNCONDITIONALLY.
```

Figure 4.5 Loading the overlay root programs

.OVOPN

Open overlays for reading

Before you can call an overlay in either a single or multitask environment, you must open the overlay file on a channel. (The same rule applies to virtual overlays in a mapped system, as a later section explains.) Several users can open an overlay file simultaneously, on different channels. The .CLOSE command closes the channel on which an overlay file has been opened.

Required Input

AC0 - Byte pointer to the name of the program overlay file, including its .OL extension.

Format

```
.SYSTM
.OVOPN n
error return
normal return
```

Variable n represents the channel number.

Possible Errors

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number
1	ERFNM	Illegal filename
6	EREOF	Mapped systems only: end of virtual overlay.
7	ERRPR	Mapped systems only: attempt to open a read-protected overlay node.
12	ERDLE	Nonexistent file.
21	ERUFT	Attempt to use a channel already in use.
26	ERMEM	Mapped systems only: insufficient memory to load (.OVLD or .TOVLD) virtual overlays.
30	ERFIL	File read error on virtual overlay file (mapped only), mag tape (bad tape).
40	EROVA	Mapped systems with virtual overlays only: not a contiguous file.
53	ERDSN	Nonexistent file.
57	ERLDE	Link depth exceeded.
66	ERDNI	Directory not initialized.
74	ERMPR	Address outside address space.
101	ERDTO	Disk timeout occurred.
102	ERENA	No linking allowed (N attribute).

.OVLOD

Load an overlay

This command loads an overlay into its reserved memory node using one of two methods. The first method, called *unconditional loading*, loads an overlay regardless of whether it resides in memory or not. This method guarantees a fresh copy of the overlay (but does not apply to virtual overlays). The second method, called *conditional loading*, loads an overlay only if it is not already in memory. Although the conditional request saves you time, it should be used for reentrant overlays only.

The .OVLOD command loads an overlay conditionally if you set AC1 to 0, or unconditionally if you set AC1 to —1. It is recommended that you make your overlays reentrant, or load them unconditionally if they are not.

Required Input

AC0 - Left byte contains the value of the overlay node; right byte contains the value of the overlay number. Alternatively, contains the symbolic name, if the .ENTO pseudo-op was used to create one.

AC1 - Input 0 to load conditionally, or —1 to load unconditionally.

Format

```
.SYSTM
.OVLOD n
error return
normal return
```

Variable n represents the channel number. Note that only one task may issue .OVLOD in a multitask environment, and that, under certain conditions (such as a nonmatching save and overlay file), the left byte of AC2 may be nonzero on an error return.

Possible Errors

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
6	EREOF	End of file.
7	ERRPR	Attempt to read a read-protected file.
15	ERFOP	File not opened.
30	ERFIL	Read error (tape).
37	EROVN	Illegal overlay number.
40	EROVA	Overlay file is not a contiguous file.
74	ERMPR	Address outside address space.
101	ERDTO	Disk timeout occurred.

.OVRP

Replace overlays in an overlay file

Although the RLDR utility can create an overlay file, it cannot modify one. You can, however, create a replacement for an overlay file with the CLI's OVLDR command, and execute the replacement with system call .OVRP or CLI command REPLACE.

With the OVLDR facility, you create a new overlay file, make the desired changes, and assign to this file the same name as the overlay it will replace. The CLI appends the extension .OR to this name. The original file is not affected by the execution of the OVLDR command; it remains the current overlay file until you execute either the .OVRP or REPLACE commands. Even if both grounds are using the original overlay file, your program can update it via system call .OVRP without halting the programs that are using it. *RDOS/DOS Command Line Interpreter* discusses the OVLDR facility in full detail.

Required Input

AC0 - Byte pointer to the overlay replacement's filename (*savefilename*.OR).

AC1 - Byte pointer to overlay filename. (*savefilename*.OL).

Format

```
.SYSTM
.OVRP
error return
normal return
```

Possible Errors

AC2	Mnemonic	Meaning
1	ERFNM	Illegal filename.
6	EREOF	End of file.
12	ERDLE	One or both files do not exist.
27	ERSPC	Out of disk space.
53	ERDSN	Directory specifier unknown.
57	ERLDE	Link depth exceeded.
66	ERDNI	Directory not initialized.
74	ERMPR	Address outside address space.
101	ERDTO	Disk timeout.

Protecting User Memory Under Mapped RDOS

This section applies to users with mapped RDOS systems. It explains how to enable and disable protection of user memory with the .WRPR and .WREBL commands. System call .WRPR allows your programs to write-protect memory in 1K blocks. This protection, which RDOS does not provide by default, remains in force until you disable it via system call .WREBL or by executing a new program.

Write protection prevents system read calls such as .RDL or .RDB—which read from a file and write to a specified address—from writing to the protected block. It also prevents such instructions as STA from writing to protected blocks. The.WRPR command does not prevent a program from loading overlays, or swapping or chaining a new program, into the write-protected blocks.

An RDOS memory block contains 1,024 decimal (1K) words; the system allots mapped memory to programs (via the CLI's SMEM command) in 1,024-word blocks; and system call .WRPR write-protects memory accordingly, in blocks of 1,024 words. If an area defined for write-protection extends across a 1024-word boundary, RDOS write-protects both blocks.

Overlay nodes can be write-protected to enhance the integrity of user code. This operation should be performed carefully, however, since a program that inadvertantly write-protects areas other than the overlay node may be unable to run properly. The RLDR utility reserves overlay nodes in integer multiples of 400_8 words, which can aid you in aligning your write-protection.

The following example steps you through the process of write-protecting an overlay node. It assumes that you are about to bind/load a program that will have one overlay segment and include 3 overlays. Ordinarily, you would enter this command line:

```
RLDR R0 R1 R2 [A,B,C D] ⟨CR⟩
```

Instead, you begin by checking the sizes of all binaries with the Library File Editor, LFE. R0, R1, and R2 require 3600_8 words, which you round off to 4000 octal. A and B are 1000_8 words each, while C D is 1500 octal words. The loader reserves an overlay node for the largest overlay—in this case, the third one of 1500_8 words. Thus, the overlay node will be 2000_8 words in length, since 1500_8 exceeds $3*400_8$. Because 2000 octal words convert to 1,024 in decimal, this size dovetails perfectly with one block of memory. As a result, you need only write-protect one memory block for this overlay node, provided you align it properly.

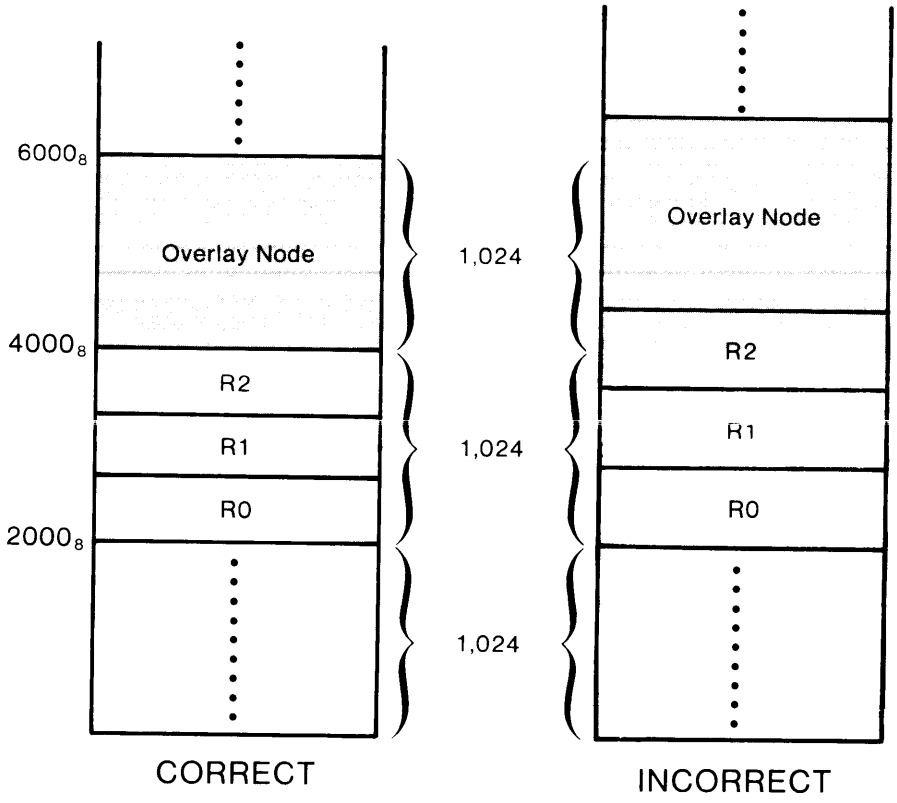
You align the node for future write protection by judicious use of the RLDR program's local /N switch. The following command line demonstrates its use:

```
RLDR R0 2000/N R1 R2 4000/N [A,B,CD] ⟨CR⟩
```

The /N switch forces the NREL pointer to the specified octal value. RLDR builds NREL upward from the bottom of user space for each binary loaded. The NREL figure pertains to the file whose name *follows* the switch. (*RDOS/DOS Command Line Interpreter* explains the use of the /N switch in full detail.) As a result of this command line, locations 4000_8 through 6000_8 are reserved for the overlay node.

Figure 4.6 shows the correct and incorrect alignment of the overlay node in memory. RO contains enough room to insert the .WRPR instruction that will protect this node without affecting the rest of the save file, for example:

```
LDA 0,LA        ;THE LOWER ADDRESS
LDA 1,HA          ;THE HIGHER ADDRESS
.SYSTM
.WRPR
JMP ER
.
.
.
LA:4000
HA:5777
```



Shading indicates memory protection.

Figure 4.6 Write-protecting memory

SD-00499

.WRPR

Protect a memory area from modification

RDOS write-enables all memory blocks by default. This system call write-protects contiguous sections of mapped memory as specified in AC0 and AC1. The blocks you specify remain protected until (1) your program disables this protection via the .WREBL command; (2) your program issues system call .EXEC, .RTN, or .ERTN; or (3) you enter a keyboard interrupt. RDOS protects memory in 1024_{10}-word blocks, just as it allocates mapped memory in blocks of 1024 decimal words. If the addresses you specify cross a block boundary, RDOS write-protects both blocks in their entirety.

Required Input

AC0 - Lower address of the series to be protected.

AC0 - Higher address of the series to be protected.

Format

```
.SYSTM
.WRPR
error return
normal return
```

The .WRPR command is a no-op in unmapped systems, and takes the normal return.

Possible Errors

This command has only one possible error: its mnemonic is ERMPR, meaning illegal address, and RDOS returns error code 74 in AC2 when it occurs.

.WREBL

Remove the write protection from a protected memory area

This system call removes the write-protect restriction from one or more blocks of memory. It write-enables memory in 1024_{10}-word blocks, just as the .WRPR command protects blocks of 1024_{10} words. Thus, if the addresses you specify cross a block boundary, RDOS write-enables all addresses in both blocks.

Required Input

AC0 - Lower address of the series to be write-enabled.

AC1 - Higher address of the series to be write-enabled.

Format

```
.SYSTM
.WREBL
error return
normal return
```

Possible Errors

This command has only one possible error: its mnemonic is ERMPR, meaning illegal address, and RDOS returns error code 74 in AC2 when it occurs.

Virtual Overlays

This section applies to users with mapped RDOS systems. It explains how to incorporate virtual overlays in your programs as a means of using extended address space. The major difference between user and virtual overlays is that the former are disk-resident, permitting only one memory-resident overlay at a time from any segment, while all virtual overlays reside simultaneously in extended address space.

After you build a virtual overlay file into your program, your program handles it as a conventional overlay file. That is, user and virtual overlays are contained in the overlay (.OL) file, which must be opened via the .OVOPN command before you can access any overlay within it. And each virtual overlay, like each conventional one, must be loaded via the .OVLOD command (or system call .TOVLD, described in Chapter5) before your program can use it. Multiple tasks may share a virtual overlay reentrantly; when all tasks have released the overlay (via system call .OVREL, in Chapter 5), another task can use the overlay node. Virtual overlays load more quickly than conventional ones because only a memory remap operation—rather than a disc access—is required. Note that you cannot "refresh" virtual overlays by reloading them. Virtual overlays are defined with the /V switch in an RLDR command line, as follows:

```
RLDR root program...[virtual overlay,...]/V
```

Virtual overlays must precede conventional ones in the RLDR command line. Space for each virtual overlay is allocated in 1K-word ($1{,}024_{10}$) pages. The loader program pads unused space. Each page begins on a 1K boundary (from page 0).

The virtual overlay node always occupies logical address space. It holds the first virtual overlay in the RLDR command line when you open the overlay file. Other virtual overlays occupy extended address space. When the program loads another virtual overlay, the new one remaps into logical space, while the original remaps into extended space. Thus, the amount of extended space that RDOS uses for each virtual overlay segment equals the node size multiplied by the number of virtual overlays in segment 1.

The following example steps through the procedures of loading and remapping virtual overlays. It is premised on the RLDR command line

```
RLDR MAIN [VW,X,Y,Z]/V [A,B,C]
```

which creates save file MAIN.SV and overlay file MAIN.OL. MAIN.OL contains binaries A, B, and C as conventional overlays, and VW,X, Y, and Z as virtual overlays.

When MAIN opens the overlay file, RDOS uses the map to set up a pointer from the virtual node to overlay VW. The .OVOPN command allocates extended memory to virtual overlays and loads them from disk into this area. Meanwhile, RDOS ignores the conventional overlay node. Figure 4.7 shows the structure of memory and disk at this time.

Next, assume that program MAIN has opened overlays on channel 3; has used virtual overlay VW; and required the use of virtual overlay Z. The new overlay is loaded as follows:

```
.
.
LDA 0,OVZ       ;OVZ WAS ASSIGNED
                ;VIA .ENTO.
SUB 1,1         ;VIRTUAL OVERLAYS ARE
.SYSTM          ;ALWAYS LOADED
                ;CONDITIONALLY.
.OVLOD 3        ;LOAD VIRTUAL OVERLAY Z.
.
.
```

As a result of this step, VW remaps into extended memory and Z remaps into logical address space, as shown in Figure 4.8.

Remember that virtual overlays are page-buffered, and that RDOS uses the largest one to determine the overlay size. Thus, overlays should be roughly equal in length or use of memory will be inefficient.

Also note that virtual overlays release extended address space only when the program performs a program swap, chain, or return. For this reason, a program that has opened virtual overlays should close them before swapping and reopen them when it returns.

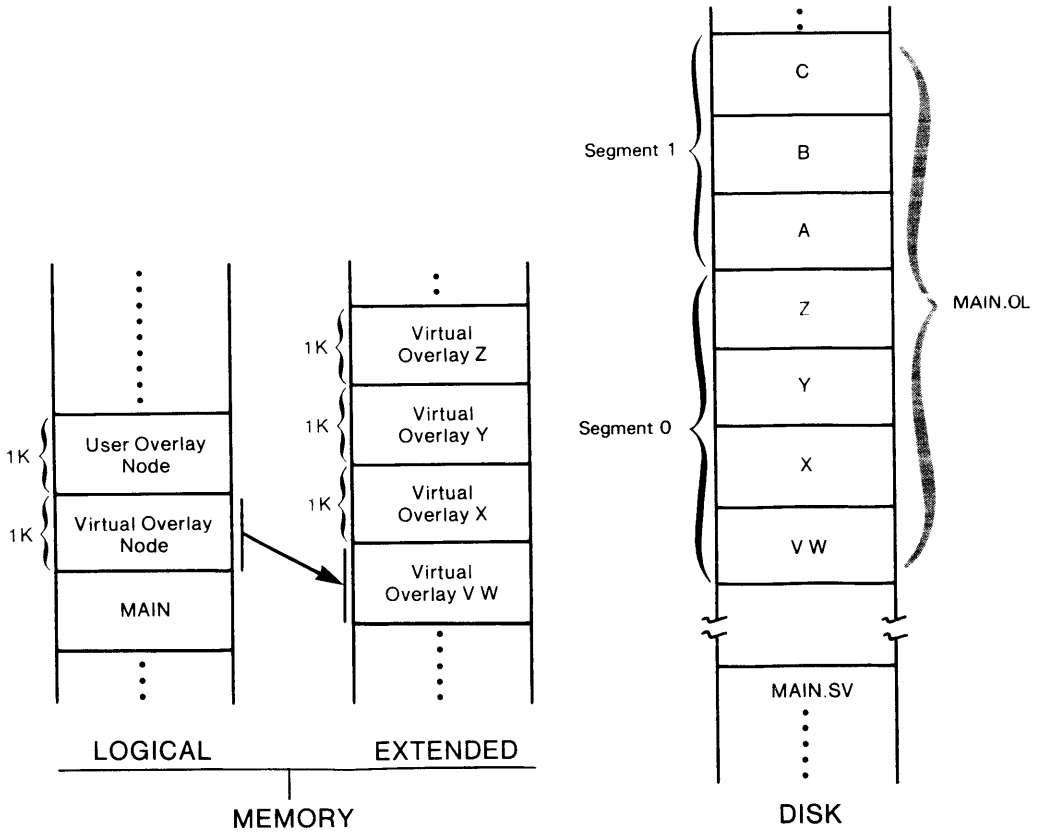


Figure 4.7 Virtual overlays before .OVLD

SD-00509

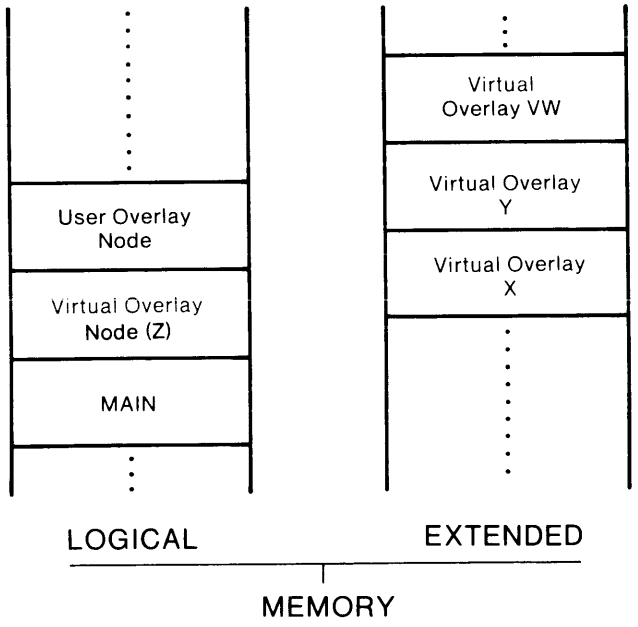


Figure 4.8 Virtual overlays after .OVLD

SD-00506

Window Mapping

Swaps, chains, and overlays help you write large programs that can run in limited amounts of address space. If your main program requires more logical memory than the computer provides, RDOS offers a different solution: window mapping.

Window mapping applies to mapped systems only. It permits direct access to portions of extended memory and allows you to transfer blocks between extended memory and disk. Both virtual overlays and window mapping can be incorporated in one program. To use a window map, follow these steps:

1. Determine the amount of memory available for extended addressing. The .VMEM command, described later in this section, enables you to do so. Check the available memory after .OVOPN and all .MEMI operations.

2. Define the size and position of the window in user address space, along with the number of blocks in the extended map. System call .MAPDF, discussed later in this section, performs these functions.

3. Logically transfer data between the window and extended memory by activating the memory management unit. Task call .REMAP, described later in this section, is used for this purpose. (Note that no true data transfer occurs; a remap operation changes the address of the data.)

After defining the map, your program can repeat the .REMAP operation as often as needed. The command should not be issued, however, when another task is using the window for I/O; .ERDB and .EWRB, the extended read/write block calls described later in this chapter, are permissible, but a task will mistakenly access the new window if other calls are issued during this time.

A program can also redefine the window, but may have only one window and one window map at a time.

Windows, like virtual overlays, are defined in multiples of 1024-word pages; they are also page-aligned. Your program accesses data in extended space by redefining the start of the window in logical address space. RDOS returns window blocks (allocated via the .MAPDF command) to the pool only when a program executes a swap, chain, or return. If your program performs a swap, the window goes away and the program must redefine it. Note that after a break or trap the state of the window in the break file is indeterminate.

Defining a Window Map

The following example demonstrates the use of the .VMEM and .MAPDF commands in defining a window map. You may want to refer to the descirptions of these commands, later is this section, to aid your understanding of how they are used here.

The example assumes that you want to define a window of 2K in logical space, with a total of 10 blocks in extended address space. Considering the rest of your program, you decide to start the window at 20000_8; it will end at 23777_8. The following sequence defines this map:

```
.
.SYSTM
.VMEM           ;ALWAYS CHECK THE NUMBER
                ;OF EXTENDED BLOCKS
                AVAILABLE. THIS CODE
                ;GIVES THE PROGRAM
                ;AN OPTION IF, FOR
                ;WHATEVER REASON, THE
                ;REQUIRED NUMBER OF
                ;16K BLOCKS ARE
                ;UNAVAILABLE.
LDA 0,C10       ;TOTAL SIZE OF WINDOW (2
                ;BLOCKS IN LOGICAL SPACE,
                ;10 TOTAL IN EXTENDED
                ;SPACE).
LDA 1,C8        ;BOTTOM OF WINDOW AT 2000 =
                ;RELATIVE LOGICAL BLOCK.
                ;SPECIFY 2 BLOCKS
LDA 2,C2        ;IN AC2.
.SYSTM          ;DEFINE THE MAP.
.MAPDF
JMP ER
.
.
.
C10: 10.
C8: 8.
C2: 2
```

Figure 4.9 shows what logical and extended memory look like as a result of this sequence.

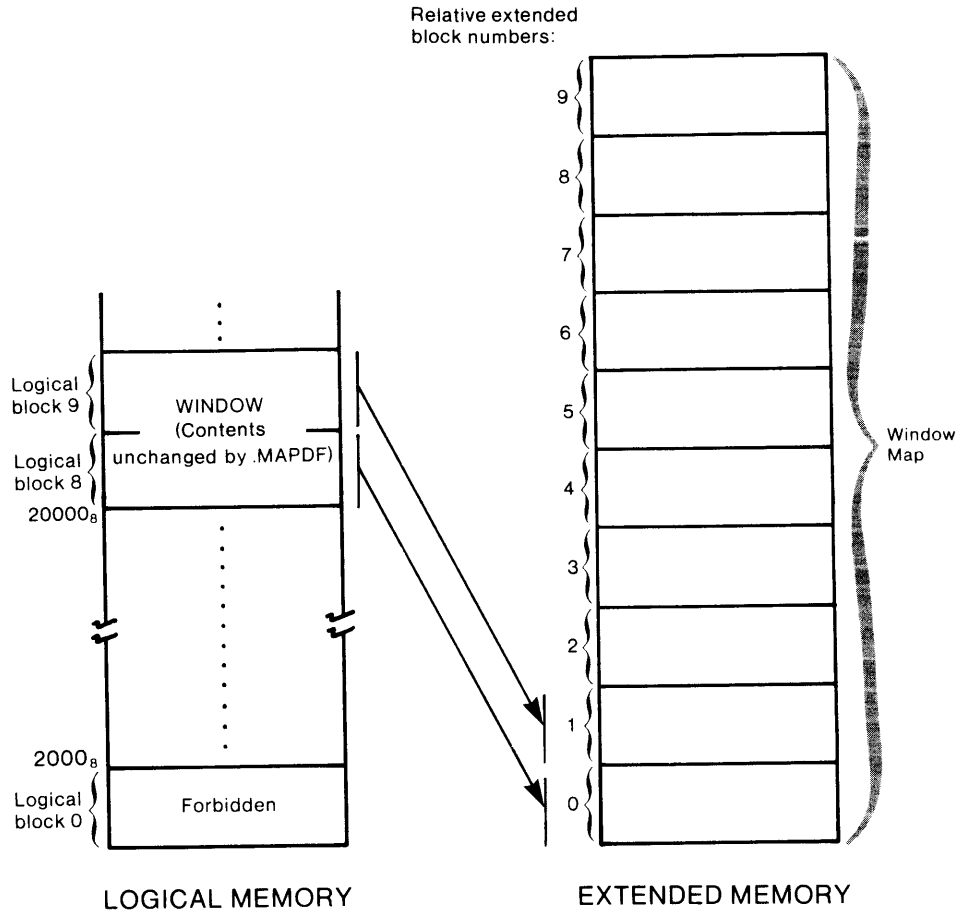


Figure 4.9 Defining a window map

SD-00507

Performing a Remap

The following example demonstrates a remap operation. It is based on the two-block window and ten-block window map discussed previously and shown in Figure 4.10. The blocks now occupying this window have become relative block numbers 0 and 1. In the coming example, a program using the .REMAP command will instruct RDOS to remap relative blocks 2 and 3 from the extended address area into the logical window.

```
.EXTN   .REMAP
.
.                       ;THE CODE IN FIGURE 4.8 IS
.                       ;IN HERE.
.
.LDA 1,BLK2             ;PUT 1ST BLOCK NUMBER(S)
                        ;TO BE REMAPPED IN LEFT BYTE
                        ;OF AC1. PUT 1ST BLOCK
                        ;IN WINDOW INTO RIGHT BYTE
                        ;OF AC1. AC1 NOW CONTAINS
                        ;THE CORRECT DATA IN
                        ;EACH BYTE FOR THE REMAP.
LDA 2,C2                ;SPECIFY THE NUMBER OF
                        ;BLOCKS TO BE REMAPPED
                        ;IN AC2 (2).
.REMAP                  ;PERFORM THE REMAP.
.
.
.
BLK2: 2B7+0B15
C2: 2
```

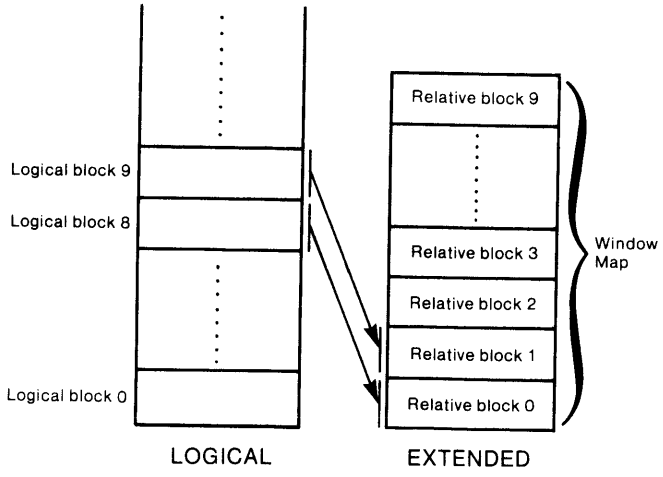


Figure 4.10 Memory before remap DG-25463

The remap occurs with little system overhead because RDOS does not actually transfer data between memory locations; rather, it simply updates the map of the memory management unit and then triggers that map. As mentioned earlier, the .REMAP command should not be issued when another task has I/O outstanding to or from a window, or this task's I/O will reference the new window.

This sequence remapped two blocks, relative block numbers 2 and 3, into the window. Alternatively, either of the blocks could have been mapped independently. Figure 4.11 shows the results of the remap operation.

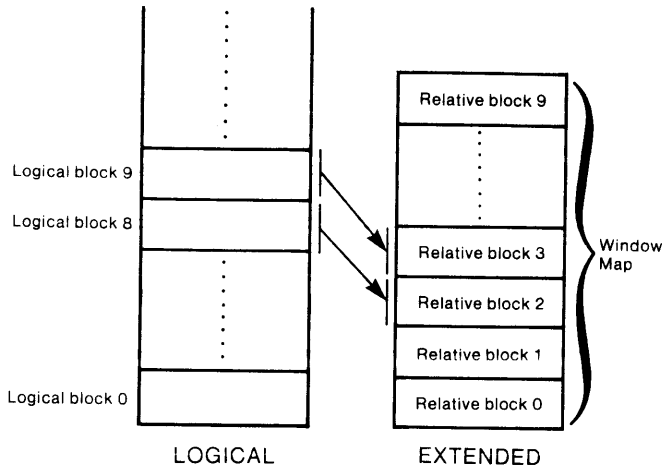


Figure 4.11 Remapping SD-00508

.VMEM

Determine the number of free blocks

The CLI's SMEM command allocates your address space. System call .VMEM provides a count of the number of free blocks available to your program for extended map use. If too few blocks are free for your program, you can change memory allotments via the SMEM command.

Required Input

None. AC0 returns the number or free memory blocks for this program.

Format

```
.SYSTM
.VMEM
error return
normal return
```

Possible Errors

None in a mapped system.

.MAPDF

Define a window and window map

As described earlier, window mapping allows your program to transfer data between a window area within logical address space and a series of blocks in extended address space. An extended or window map contains a list of physical memory blocks that can be mapped into the window. System call .MAPDF defines a window and window map; only one window and map can exist within a program. You must define the window area in the address space below NMAX.

The .MAPDF command assigns relative extended block numbers 0 to n-1 to the blocks in extended memory, where n equals the number specified in AC0. The first window block in logical space receives extended relative block number 0, the second block (if any) in logical space receives number 1, and so forth; the numbers proceed sequentially in extended space, as Figure 4.9 showed earlier. Note that defining the window map does not alter the initial contents of the window.

Required Input

AC0 - Total number of memory blocks to be assigned to the extended memory area. (This number includes any blocks in logical address space that currently reside within the window.)

AC1 - The starting page number for the window in logical space, from 1 through 31_{10} (or 1 through 30 for NOVA 830 and 840 computers). The first block, number 0, cannot be specified in AC1 because it includes page zero. Remember that a window is block-aligned, causing its logical starting address to coincide with the start of a block.

AC2 - The size of the window in 1K blocks.

Format

```
.SYSTM
.MAPDF
error return
normal return
```

Possible Errors

Only one possible error results from this command: its mnemonic is ERMEM, indicating that the specified block is out of the window map's range. RDOS returns error code 26 in AC2 as a result.

.REMAP

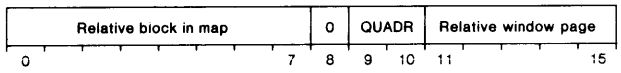
Perform a logical window transfer

Once your program has defined a window and window map, it can remap data from the memory in the window map to or from any part of the window in logical space. The .REMAP command performs a remap operation by placing blocks from the extended address area into the window.

On machines with a memory expansion option, user programs can access main memory above the 256th 1KW page. If you chose the option of a shared data area during system generation, the .REMAP command allows you access to it. In systems that incorporate an array processor, array processor memory can be mapped into the user program window. Note that .REMAP may be issued from either ground; if both grounds issue this call, the same address can be allocated to both. However, interground communication via the array processor is not supported. Also note that the .REMAP command is a task call, requiring that you specify its name in an .EXTN statement.

Required Input

AC1: - Window position in MAP table as follows:



The left byte of AC1 contains the starting relative block number in the map. Pass the starting, relative block number of the array processor if you have this feature and plan to use window mapping in conjunction with it.

The right byte of AC1 contains an extension field, QUADR, and the relative window page. For all except the ZRDOS version of mapped RDOS, bits 8 through 10 must be zero. For ZRDOS, bit 8 must be zero, while bits 9 and 10 select the desired 256-page segment in the program's extended memory mapping table.

Bits 9 and 10 may be nonzero only if the program will run on a mapped ECLIPSE system having more than 256 KW of main memory, and that program has reserved more than 256 one-KW memory pages.

Bits 11 through 15 contain the relative window page number.

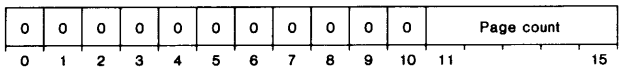
Main Memory access extension field, "QUADR" - "ZRDOS" only

QUADR Relative position (page) in MAP table

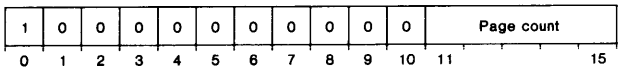
	0 – 255 decimal	(0 – 377 octal)
00		
01	256 – 511	(400 – 777)
10	512 – 767	(1000 – 1377)
11	768 – 1023	(1400 – 1777)

AC2 - The call parameters passed in AC2 determine into which area of extended memory the user program's mapping window is to be remapped, along with the number of pages to be remapped. The window may be mapped into the program's own unshared, extended memory space; the shared data pages; or the array processor memory in systems that include array processor hardware. The option of shared data pages is available with all mapped RDOS systems and is selected during system generation.

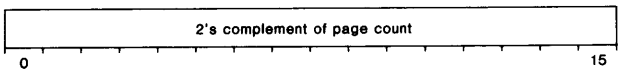
For unshared, extended memory access, the parameters passed in AC2 are:



For shared, extended memory access, the parameters passed in AC2 are:



For array processor memory access, the parameters passed in AC2 are:



Note that all bits indicated as set to zero must be zero for proper operation of the .REMAP system call. Also note that block numbers within the windows and map are relative numbers beginning with 0.

Format

```
.REMAP
error return
normal return
```

The contents of all accumulators are lost upon return from this call.

Possible Errors

AC2	Mnemonic	Meaning
32	ERADR	Illegal starting address.
36	ERDNM	The array processor was not specified during system generation.
26	ERMEM	Insufficient memory; attempt to .REMAP past the end of memory.

Extended Direct Block I/O

After your program has defined a window map, it can use extended, direct block I/O—a special form of I/O similar in concept and operation to direct block I/O, which transfers 256-word blocks between core and disk without using an intermediary system buffer.

The extended direct block I/O commands .ERDB and .EWRB apply to mapped RDOS systems only and are the subject of this section. Descriptions of these commands are followed by an example illustrating their use. Both system calls transfer 256-word data blocks between the map in extended memory and disk files. This I/O type provides a quick means of altering data in the extended memory area. Your reference is independent of any remaps that have occurred or may occur during the execution of these calls. Moreover, this form of I/O transfers disk file data directly to the extended memory area, without passing it through the window in logical address space.

Neither the .ERDB or .EWRB commands use an intermediary buffer, and their calling sequences resemble those of the direct block I/O commands (.RDB and .WRB) discussed in Chapter 3.

Note that the .ERDB and .EWRB commands are restricted from access to the shared data area, if you reserved one during system generation.

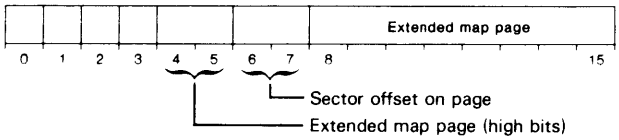
.ERDB

Extended read direct block

This system call reads up to 128 disk blocks (256 words each) from a randomly or contiguously organized file into one or more 1024-word extended memory pages. The command resembles its direct block I/O counterpart, system call .RDB, except for the parameter passed in AC0. Because .ERDB reads into extended memory rather than directly addressable, logical memory, the parameter you pass in AC0 specifies the map's relative memory page number (in the range 0—1023), and a 256-word offset into this page. Since you must have defined a window map in order to issue this command, you should know the relative block numbers in the map.

Required Input

AC0 - Contains the extended map page number, the block offset into the page, and the high order bits of the page number. The bits are organized as shown below:



The following table shows the organization of bits six and seven, which point to one of the four page sectors, which we call quarter block sectors.

Contents of bits 6-7	Quarter block sector ranges (octal)
00	0 – 377
01	400 – 777
10	1000 – 1377
11	1400 – 1777

AC1 - The starting, relative disk block number in the file from 0 to n-1 for a file consisting of n disk blocks.

AC2 - Left byte specifies the number of 256-word disk blocks to be read. Right byte specifies the channel number, if file was opened on channel 77.

Format

```
.SYSTM
.ERDB n
error return
normal return
```

Variable n signifies a read from the disk file opened on channel n (or 77).

Possible Errors

AC2	Mnemonic	Meaning
0	ERENO	Illegal channel number.
3	ERICD	Illegal command for device.
4	ERSVI	Not a randomly or contiguously organized file.
6	EREOF*	End of file.
7	ERRPR	File is read-protected.
15	ERFOP	No file is open on this channel.
30	ERFIL	File read error (on magnetic tape, signalling a bad tape).
40	EROVA	File not accessible by direct block I/O.
74	ERMPR	Address outside address space.
101	ERDTO	Disk timeout occurred.

*Upon detection of error EREOF, RDOS returns code 6 in the right byte of AC2; the left byte contains the partial read count.

.EWRB

Extended direct write block

This command writes up to 128, 256-word blocks from extended memory to a randomly or contiguously organized disk file. The current contents of the window remain unchanged, as do the contents of the map.

The .EWRB call resembles its direct block I/O counterpart, system call .WRB, except for the parameter passed in AC0. This parameter must indicate both the relative, extended memory block number (in the range 0—244) and a 256-word offset into this block. Your program must have defined a map via the .MAPDF command before issuing .EWRB; thus, you should know the relative, 1K block numbers in the map. For details on the offset, see the previous discussion of system call .ERDB.

Required Input

AC0 - Right byte specifies the extended memory block number. Left byte specifies a write from the first 256-word group if set to 0; a write from the second 256-word group if set to 1; a write from the third 256-word group if set to 2; or a write from the fourth group of 256 words if set to 3.

AC1 - Start writing to this relative block number in the disk file.

AC2 - Left byte specifies the number of 256-word blocks to be written. Right byte specifies the channel, if file was opened on channel 77.

Format

```
.SYSTM
.EWRB n
error return
normal return
```

Variable n signifies a write to the disk file opened on channel n (or 77).

Possible Errors

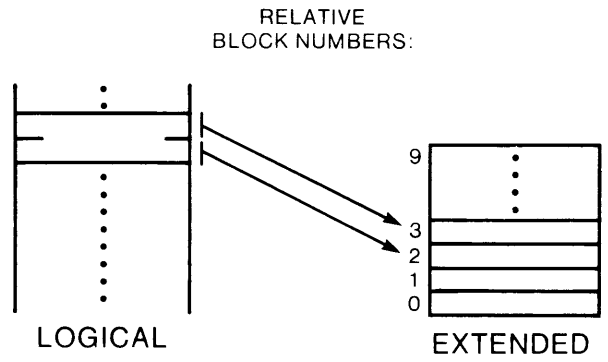
AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
3	ERICD	Illegal command for device.
4	ERSVI	Not a randomly or contiguously organized file.
6	EREOF	End of file in a contiguous file.
10	ERWPR	File is write-protected.
15	ERFOP	No file is opened on this channel.
27	ERSPC*	Disk space is exhausted.
40	EROVA	File not accessible by direct block I/O.
74	ERMPR	Address outside address space.
101	ERDTO	Disk timeout occurred.

*Upon detection of error ERSPC, RDOS returns code 27 in the right byte of AC2; the left byte contains the partial write count.

If you write to a sequential or random file and get ERSPC, you must delete the file in order to recover the disk space allocated to the file before the error occurred. You must do this even though the CLI LIST command may show a zero file length.

Extended Direct Block I/O Example

To conclude the discussion of extended, direct block I/O, Figure 4.12 demonstrates the use of the .ERDB command to transfer a disk file to the map. This figure continues the example of Figures 4.10 and 4.11, in which the .REMAP command was used to remap relative blocks 2 and 3 their extended address area into the logical window. Now, instructed by the code in Figure 4.12, RDOS writes file E to relative block number 0 and 1 in the map.




```
LDA 0, FILEE      ;BYTE POINTER TO DISK FILE E.
SUB 1, 1          ;DEFAULT DISABLE MASK.
  .SYSTM
  .OPEN 3         ;OPEN E ON CHANNEL 3.
    •             ;
    •
  SUB 0, 0        ;GET 0 TO START READING
                  ;TO EXTENDED BLOCK 0.
  SUB 1, 1        ;GET 0 TO START READING
                  ;FROM STARTING POSITION IN
                  ;DISK FILE E.
  LDA 2, C8       ;SPECIFY THE NUMBER OF DISK
                  ;BLOCKS TO BE READ, IN LEFT
                  ;BYTE OF AC2 -- 8. THESE
                  ;BLOCKS WILL FILL MAP BLOCKS
                  ;0 AND 1.
  .SYSTM
  .ERDB 3         ;READ FROM FILE E ON CHANNEL 3.
    •
    •
    •
  C8: 8.B7
FILEE:.x1*2
      .TXT "E"
```

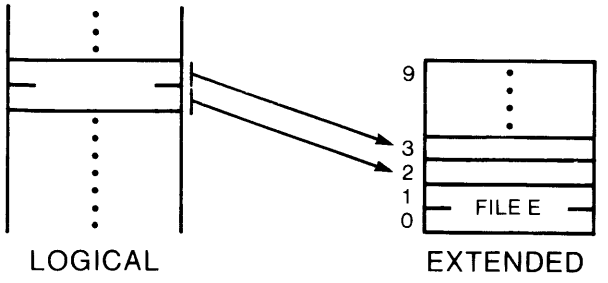


Figure 4.12 Extended block read SD-00746

The 256-word offset into the selected block will indicate either 0, 400_8, 1000_8, or 1400_8, for the start of each 256-word disk block. RDOS adds the extended memory block number and the offset. This produces the memory block number in the right byte and either 0, 1, 2, or 3 in the left byte for the first, second, third, or fourth 256-word disk block.

Summary

This section summarizes all system (and task) calls discussed in this chapter in Table 4.1.

System Call	Function
.ERDB	Read data blocks from disk into extended memory.
.ERTN	Return from a program swap with the error status of the calling program.
.EWRB	Write blocks from extended memory to disk.
.EXEC	Swap or chain in a new program.
.MAPDF	Define a window and window map.
.OVLOD	Load an overlay into the area of memory reserved for it.
.OVOPN	Open an overlay file on a specified channel.
.OVRP	Replace the overlays in an overlay file.
.REMAP	Activate a logical window transfer.
.RTN	Return from a program to a higher-level program.
.VMEM	Determine the number of memory blocks available for extended addressing.
.WREBL	Remove the write-protection from a protected area of memory.
.WRPR	Protect an area of memory from modification.

Table 4.1 System and task call summary

Chapter 5

Multitask Programming

This chapter describes tasks, task management, task overlay management, and task control from the console via operator messages. It begins by explaining task prioritites and the Task Control Block, which the RDOS Task Scheduler uses to keep track of each task in a program. Then the possible task states and the User Status Table, which monitors all TCBs during program execution, are described. The remaining sections discuss the commands that task may issue to control itself or other tasks. In order of appearance, these sections are:

- Task Initiation
- Task Termination
- Task State Modification
- Intertask Communication
- Overlay Management
- Enqueuing Tasks
- User/System Clock Commands
- Task Management by ID Number
- Task/Operator Communications
- Task/Operator Communications Module (OPCOM)
- Disabling and Enabling the Multitask Environment
- Disabling and Enabling the Task Scheduler

The task and system calls discussed in each section are summarized at the end of the chapter in table form.

Your program is the initial task in a multitasking environment. After it initiates one or more tasks, any of those tasks may issue a task or system call. It is recommended that you assign an ID to each task routine that you write. Although not mandatory, an ID number is required by certain useful task calls and by OPCOM, the console communications feature described later in this chapter.

Your program initiates a task via the .TASK or .QTSK commands or, from the console, via OPCOM commands RUN or QUE. RDOS then assigns the task a TCB from the TCB pool that you establish via a .COMM TASK statement or during loading. The task is then ready for execution. Depending on its priority and other conditions specified in your program, the task achieves CPU control and executes. It retains control of the CPU until it suspends itself, or until it is suspended by a task of equal or higher priority that has requested the CPU's services after an interrupt.

When suspended, the task's TCB saves its current state. The program's User Status Table monitors all TCBs and their associated tasks, enabling the Task Scheduler to resume execution of any suspended task from the point of suspension. The task retains its TCB until it is killed (or kills itself) via the .KILL, .AKILL, .ABORT, or .OVKIL commands, or via the OPCOM command KIL. After a task has been killed, its TCB returns to the free TCB pool. The task remains inert until you reinitiate it, when it receives another TCB.

Each task that you include in your program is memory-resident during program execution unless it resides in an overlay. If a task resides in an overlay, the program must open the overlay with system call .OVOPN and load it via the .TOVLD or .QTSK commands discussed later in this chapter.

Task Priorities

Task priorities range from 0 through 255 decimal, where 0 is the highest priority. RDOS automatically assigns priority 0 for the task whose starting address you specify in the .END statement at the end of your program.

Several tasks may exist at the same priority. Tasks of equal priority receive CPU control on a round-robin basis, in which the task that most recently received control will be the last to receive it again, unless other tasks are unable to assume control when they are scheduled to do so. When your program changes a task's priority via the .PRI command, RDOS places this task at the end of a list of all other tasks that share its new priority.

Task Control Blocks

A task is an asynchronous, execution path through user address space that demands the use of system resources. You can assign many tasks to a single reentrant path, and you can assign each of these tasks a unique priority. Given

the asynchronous nature of tasks, the RDOS Task Scheduler must maintain information about the status of each. RDOS retains status information within a Task Control Block (TCB); there is one TCB for each task. Table 5.1 describes the structure of TCBs. The text that follows expands on the information in this table.

Word	Mnemonic	Contents
0	TPC	User PC (B0-14) and Carry (B15)
1	TAC0	AC0
2	TAC1	AC1
3	TAC2	AC2
4	TAC3	AC3
5	TPRST	Status bits and priority
6	TSYS	System call word
7	TLNK	Link word, to next TCB
10	TUSP	USP (User Stack Pointer)
11	TELN	Extended save area
12	TID	Task ID number, right byte
13	TTMP	Temporary storage area for Scheduler
14	TKLAD	Task kill address, if program specified one
15	TSP	Stack pointer
16	TFP	Frame pointer
17	TSL	Stack limit
20	TSO	Overflow address, for single task environment

Table 5.1 Structure of a task control block (TCB)

Words 1 through 4 of the TCB structure are self-explanatory. Figure 5.1 diagrams the task state and priority information contained in word 5, TPRST.

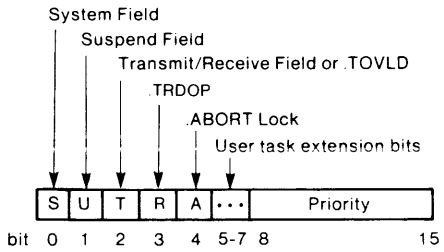


Figure 5.1 Task state/priority information (TPRST) SD-00541

Note that word TPRST divides into five fields, associated with letters S, U, T, R, and A. The Task Scheduler sets these fields as follows:

Field	Bit Setting/Meaning
S	1 = Task has issued a system call and has been suspended until the call has executed. 0 = System call has finished executing, or no call is outstanding for the task.
U	1 = Task is suspended by a .SUSP, .ASUSP, or .TIDS command.
T	1 = Task has issued either the .XMTW and .REC commands or call .TOVLD.
R	1 = Task is awaiting a message via the .TRDOP command.
A	1 = Task is in the process of aborting.

As shown in Figure 5.1, the remainder of word TPRST contains a reserved bit 5; two extension bits (6 and 7, or TSUPN and TSUSR). which allow you to expand the RDOS task-handling mechanism as described in Appendix I; and bits 8 through 15, which hold the task's priority.

RDOS uses word 6 of the Task Control Block, TSYS, to store information about system calls and the .XMTW, .REC, and .TOVLD commands.

Word 7, TLNK, contains the starting address of the next TCB in the chain.

Word 10, TUSP, contains the value of location USP at the time this task last changed from the executing state. You may use USP as a general-purpose storage location for each task while it is executing. The system restores the value of USP for each task that gains control of the CPU.

Word 11, TELN, points to the task's highest-level language save area; if you do not use it, the system sets TELN to 0.

Word 12, TID, contains the task identification number, if any, in its right byte.

Word 14, TKLAD, contains the address that will receive control whenever a task is killed, provided you have defined such an address via the .KILAD command. Bit 0 is set if a .KILL or .ABORT command has been issued for this task.

Words 15 through 20 of the TCB contain save information pertaining to a stack's state. This information is reserved for TCBs.

Building Multitask Programs

Before running a multitask program, you must specify both the number of RDOS channels and the number of TCBs that this program needs. You can do this before assembly, within the program, via a .COMM TASK statement. You can also specify tasks and channels with the local /K and /C switches in an RLDR command line.

A .COMM TASK statement must appear in the first binary of the RLDR command line, since it affects the loading process of the remaining program and determines which task scheduler (TMIN or TCBMON) will become a part of it. When the /C or /K switches are used in conjunction with a .COMM TASK statement, the switch information overrides that of the statment. The format of source program statements is:

```
.COMM TASK, k*400 + c
```

where k represents the octal number of tasks, and c represents the octal number of RDOS channels that your program will use, for example:

```
.COMM TASK, 7*400 + 16
```

In mapped systems, the maximum number of tasks (k) cannot exceed 44_{10}. This is due to the requirement, in mapped systems, that all TCBs reside in NREL, in the first, 1K-page of memory. If the program uses overlays, the overlay directory must also reside in the first 1K-page, which reduces space for TCBs.

Data General supplies task schedulers TMIN and TCBMON, all task command modules, and the interrupt-on symbolic debugger in the system library, SYS.LB. Unless you specify otherwise with an RLDR switch, the loader program places all items required from the library directly above the program code.

NOTE: *Because the system library differs for each type of system (eg, unmapped and mapped NOVA), programs loaded under one type of system may not execute under another type. To load for a different kind of system, you must obtain the proper system library for it and ensure that RLDR searches for this library, rather than the current one, during the loading process. You can do this by loading from a subdirectory that contains the target system's library and links to RLDR.*

To write your own task command modules or define a task memory or FPU save area, you can refer to the source listings for the system library if you acquired them with your system.

Conserving ZREL Space

Normally, each unique task call in your program requires one word of page zero (ZREL) space. Note, for example, the conventional use of the .TASK command:

```
.EXTN .TASK
.
.               ;SET UP ACCUMULATORS.
.TASK
.
```

In this example the task call word (.TASK) is resolved by SYS.LB to a JSR instruction that transfers control through a page zero address. Thus, .TASK requires one word of ZREL. (Subsequent .TASK calls will not require additional ZREL.) Alternatively, to conserve ZREL space, replace each task call with a transfer to a label with the same name as the original task call, but with the first two characters transposed. The transfer must be a JSR or equivalent, and you must declare the transposed call in an .EXTN statement. The following example demonstrates this transposition scheme, and uses no ZREL space:

```
.EXTN T.ASK
.
.               ;SET UP ACCUMULATORS.
JSR @ TASK0
TASK0: T.ASK
```

Task States

A task can exist in any of three states: (1) it is ready to perform its functions; (2) it is actually in control of the CPU and is executing its assigned instruction path; or (3) it is suspended and temporarily unable to receive CPU control. A task can also be dormant, having relinquished its TCB (or never having had one); a dormant task has no priority and no chance of gaining CPU control until activated by a .TASK or .QTSK command. The Task Scheduler always gives CPU control to the highest priority task that is ready.

Suspended tasks are those having at least one of the four status bits (S, U, T, or R) in word TRPST set to 1. A task may become suspended for one or more of the following reasons:

- It has been suspended by an .ASUSP or .TIDS command.
- It has suspended itself for a specified period via the .DELAY command, or for an indefinite period via the .SUSP command.
- It is waiting for a message from another task via the .REC command.
- It has issued a message-and-wait command, .XMTW.
- It is waiting for the use of an overlay node.
- It has issued a system call and is waiting for it to finish executing.

Just as a number of different events can suspend a ready task, several events can ready a suspended task:

- The .ARDY or .TIDR commands can be issued for the task.
- The task message that it has been instructed to wait for via the .REC command.
- The loading of a requested overlay.
- The completion of a .SYSTM call (such as a request for I/O).

A task that is suspended by a command and by some other event must be readied by an .ARDY or .TIDR call *and* by whatever other event suspended it. Such a task is said to be doubly suspended, with bits S and U set in word TPRST of its Task Control Block. The environment must allow RDOS to reset bits S, U, T and R to ready this task.

You can delete tasks from the active queue and place them in dormancy either separately, via the .KILL, .TIDK or .ABORT commands, or by priority group, via the .AKILL command. Tasks that you have deleted add their empty TCBs to an inactive chain of free element TCBs.

If all tasks are killed and no task is awaiting execution via the .QTSK command, the effect is the same as if system call .RTN had been issued. Program control then returns to the next-higher program level.

TCB Queues

There is one TCB queue for tasks that are currently executing, suspended, or ready. This queue consists of a chain of TCBs, connected by word TLNK in each TCB, and is called the *active chain*. USTAC of the User Status Table points to the first TCB; this TCB points to the next one, and so forth. The last TCB in the chain has the value -1 in word TLNK.

A *free element chain* is a simple queue of dormant TCBs. TCBs in the free element chain are joined by TLNK words; all other words in each dormant TCB are unused. There is no priority among TCBs in this kind of chain. USTFC of the User Status Table points to the first TCB in the free element chain, as shown in Figure 5.2.

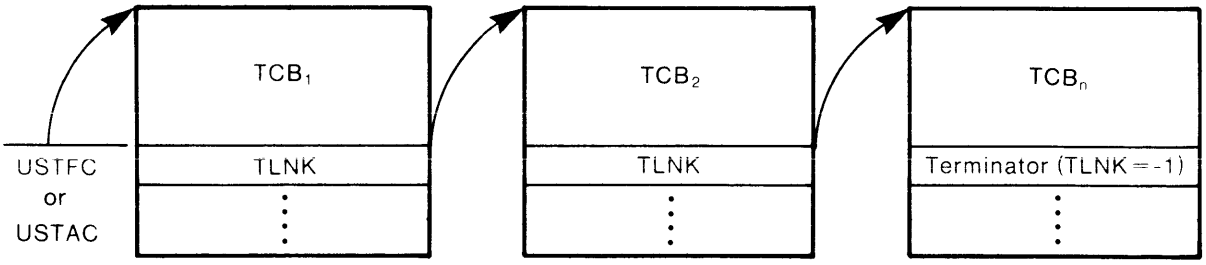


Figure 5.2 TCB chain

SD-00542

Task Synchronization and communication

Each task can communicate with another by sending a one-word message to an agreed-upon location in user address space. This address space includes all locations from address 16 through NMAX. (Avoid locations 0 through 17_8 and 40 through 47_8 in ZREL, along with system tables directly above 400_8.)

The task sending a message may either return to the Task Scheduler immediately (.XMT) or suspend itself (.XMTW) until a receiving task has issued a receive request (.REC) and has received the message. Receipt of the message includes resetting the contents of the message location to zero. Upon receipt of the message, its recipient has bit T set to 0. The message location must contain 0 before the message is sent.

User Status Table

The User Status Table (UST) is a 24_8 word table that records runtime information about a program. This table is located at addresses 0400_8 through 0423_8. Table 5.2 shows the structure of the UST in memory; the contents of each address are expanded on in the text that follows.

Address	Label	Contents
012	USTP	ZREL pointer to UST*
400	USTPC	Used by the system
401	USTZM	ZMAX
402	USTSS	Start of Symbol Table (SST)
403	USTES	End of Symbol Table (EST)
404	USTNM	NMAX after runtime .MEMIs
405	USTSA	Starting address of Task Scheduler
406	USTDA	Debugger address: -1 if the debugger was not loaded
407	USTHU	USTNM after loading (original NMAX)
410	USTCS	FORTRAN common area size
411	USTIT	CTRL-A interrupt address: -1 initially
412	USTBR	CTRL-C or .BREAK address: -1 initially
413	USTCH	Number of TCBs (left byte) and channels (right byte)
414	USTCT	Current TCB pointer
415	USTAC	Start of active TCB chain
416	USTFC	Start of free TCB chain
417	USTIN	Initial start of NREL code (INMAX)
420	USTOD	Overlay directory address
421	USTSV	Available for use by the system
422	USTRV	Revision level number, and, during execution, the environment state.
423	USTIA	Address of TCB for console interrupt task: 0 initially.

Table 5.2 Structure of user status table (UST)
*The UST for a program running in an unmapped foreground starts at the beginning of the foreground memory partition.

Location 12 in page zero is USTP, which points to the start of the User Status Table belonging to the currently executing foreground or background program. The loader creates symbol USTAD as an .ENTO declaration. This symbol also points to the base of the program's UST.

Location 400, or USTPC, indicates which program is running, where 0 indicates the background program and 1 indicates the foreground program.

Location 401, or USTZM, contains ZMAX, the first free location in page zero after loading.

Locations 402 and 403, USTSS and USTES, point to the start and end of the symbol table, respectively. By default, RLDR loads the symbol table so that its last location plus one coincides with the value of NMAX. If you request that RDOS place the symbol table in upper memory (via the global /S switch in your RLDR command line), the symbol table is moved so that it will be immediately below RDOS space when the save file is executed. If the symbol table has not been loaded, locations 402 and 403 contain zeroes.

Location 404, or USTNM, contains the current value of NMAX at runtime. This value changes as NMAX is increased or decreased. Location 407, USTHU, is set by the loader to the value of NMAX after loading. RDOS never changes this word during program execution.

Location 411, USTIT, is the interrupt address (CTRL-A). After loading, this address is set to -1. If it is unchanged at runtime, control goes to the next higher-level program with USTIT set to a valid address when a CTRL-A interrupt occurs. (If the foreground is interrupted and no higher level program exists in the foreground with a valid USTIT address, RDOS terminates the foreground.) The user core image is not saved. At execution time, your program can set USTIT to an address to which the system will transfer control when a CTRL-A interrupt occurs.

Location 412, or USTBR, is the break address (CTRL-C). After loading, RDOS sets this address to -1. Whenever a CTRL-C break occurs, the system writes the core image to file BREAK.SV (or FBREAK.SV in the foreground) in the current directory. If USTBR remains unchanged at runtime, control passes to the next higher-level program with USTBR set to a valid address when a CTRL-C interrupt occurs. Alternatively, you can set USTBR to an address to which control will pass upon successful creation of the break file. If RDOS cannot create a break file (eg, because it is out of disk space), control goes to the address specified by USTBR minus one. AC2 will contain the error code.

Location 413, USTCH, contains the number of program TCBs in its left byte, and the number of I/O channels in its right byte.

Location 421, USTSV, is reserved for RDOS.

Location 422, USTRV, is reserved for storage of the revision number for this save file, and for runtime data on the machine that is running the program. Revision numbers can extend from 00 to 256; RDOS stores the major revision number in the left byte, and the minor revision number in the right byte, of this word. During a program's execution, USTRV contains values that indicate what kind of machine and RDOS system is running the program. You can find these values, along with interpretations of them in the listing of file PARU.SR contained in Appendix B. (Refer to the heading ENVIRONMENT STATUS BITS IN USTRV.) Location 423, or USTIA, of the User Status Table contains the TCB address of the task that issued an .INTAD system call. The loader initializes this word to 0.

Task and System Calls

There are four essential differences between task calls and system calls:

- Task calls have no .SYSTM mechanism. Instead, each call uses a module from the system library, requiring that you declare each task call included in a program as external via an .EXTN statement. If your program fails to declare each call external, RLDR neglects to load the call's module and the call will not work.
- RDOS executes all system calls in RDOS space, but executes all tasks calls in user space. Thus, the diversity of task calls in a program affects the program's size, while the diversity of system calls does not.
- Most task calls do not have error returns, and hence do not reserve an error return location.
- Accumulators are used to pass all parameters to most task calls. You will generally use AC0 and AC1 to enter or return data. Occasionally, AC2 is used to enter data. When an error is defined for a call, AC2 will contain the code on an error return.

On return from all task calls, AC3 contains USP, the contents of location 16_8, by default. RDOS maintains the frame pointer in location CSP. If yours is a NOVA 3 computer, you can return the contents of the hardware frame pointer in AC3 by loading the program with module N3SAC. (In NOVA 3s, the hardware stack is moved for each task swap, but the stack overflow handler remains at location 43_8.) On an ECLIPSE computer, you can return the frame pointer by inserting module ESAC3 in the RLDR command line. Returns in AC3 can be summarized as follows:

If program was loaded with module:	*Then upon return AC3 contains:*
NSAC3 (any machine; always used by default)	Contents of USP (location 16_8)
NSAC3 (NOVA 3s only)	Contents of frame pointer register
ESAC3 (ECLIPSEs only)	Contents of frame pointer (location 41_8)

In summary, task calls differ from .SYSTM calls in four ways:

1. Task calls reference library modules, and must be declared external. Task calls are not preceded by the .SYSTM mnemonic, and are resolved by the binder/loader to be JSR calls to task processing modules.

2. Task calls are processed in user address space, while RDOS or system calls require system action which occurs in RDOS space.

3. Only some task calls have error returns. Those without error returns do not reserve an error return location.

4. You must pass all parameters to task calls via the accumulators. (The .QTSK command is the only exception to this rule.)

Task Initiation

This section describes the .TASK command, which initiates any memory-resident task. The .QTSK command, described in a later section, initiates either a core-resident or overlayed task for periodic execution.

.TASK

Create a task

This command initiates a new task at a specified priority in your program, and assigns an identification number to the task if you desire. When you load the program, only one task exists; therefore your system must issue this or the .QTSK command to initiate a multitask environment.

The .TASK command passes the contents of AC2 to the created task. This permits your program to relay an initial, one-word message to the newly created task.

Required Input

AC0 - Right byte: priority of the new task, ranging from 1 to 377. If you set this byte to zero, the priority of the new task will be identical to that of the calling task. Left byte (optional but recommended): ID number for the new task, ranging from 1 to 377. You may give an ID number of zero to more than one task. Each nonzero ID must be unique.

AC1 - Address where the new task will begin execution.

Format

```
.TASK
error return
normal return
```

Possible Errors

AC2	Mnemonic	Meaning
42	ERNOT	No TCBs available.
61	ERTID	A task with the requested ID (except 0) already exists.

Task Termination

This section describes the commands your program can use to kill tasks without using their ID numbers.* In order of discussion, these commands include:

.KILAD	Define an address that will receive control when a task is killed.
.KILL	Kill the calling task.
.AKILL	Kill all tasks of the specified priority.
.ABORT	Kill the specified task and its currently executing system call, if any.

So that your program can proceed efficiently, RDOS provides the .KILAD command, which specifies an address to receive control before a task is killed. This address can instruct the task to close its channel(s), release its overlay(s), or give it a choice of action.

For most orderly terminations, or for those that occur via the .AKILL or .TIDK commands, RDOS raises each task you are terminating to the highest possible priority and readies it. If several tasks exist with a priority of 0, RDOS services them before killing the specified task(s). Thus, if a task has been suspended by the .REC, .XMTW, .SUSP, or .TIDS commands, RDOS lifts the suspension. If the task is suspended because of an outstanding system call, RDOS completes that call before readying the task. In either case, RDOS terminates the task you wish to kill when it receives control of the CPU, unless your program has specified a kill-processing address.

When you specify a kill-processing address via task call .KILAD, control passes to that address when the task gains control of the CPU. This allows the task to close any channels or release any overlays it was using. Moreover, the kill-processing routine serves as a reprieve, since RDOS does not actually terminate the routine until it is killed a second time. The kill-processing routine can thus act as a validating procedure in which it determines whether or not the target task should be terminated. At this point, the task being killed can renew its kill-processing address by reissuing the .KILAD command.

After a task has been killed by any means, it relinquishes its TCB to the free TCB pool for possible use by future tasks.

*Commands that control tasks by ID number are described in the section entitled "Task Management By ID Number" later in this chapter.

.KILAD

Define a kill-processing address

This task call permits a task to define a special address that will gain control the first time that your program tries to terminate the target task. On a second attempt to kill the task, RDOS terminates it without transferring control to the kill-processing address.

The kill address allows a task to release system resources before terminating. Each task must explicitly release such resources as overlays, channels, user devices and user clock definitions; the code that performs this function can be written into the task's .KILAD routine. After releasing these resources and following any other instructions, the task must issue a .KILL command to terminate itself. On this second attempt to terminate the task, termination occurs immediately.

Alternatively, the target task may decide not to terminate itself. In this case, before branching out of the kill-processing routine, the task should issue a .KILAD call to the same or to a different kill-processing routine. This measure ensures that a later attempt to kill this task will cause it to branch once again to its kill-processing routine.

A task in a kill-processing routine executes at the highest priority; it has CPU control. Such routines retain control until they relinquish it via a transition in task state or a change of priority level.

Required Input

AC0 - Address of the kill-processing routine.

Format

```
.KILAD
normal return
```

Possible Errors

None.

.KILL

Delete the calling task

This command deletes the calling task's TCB from the active queue and places it in the free element TCB chain. The calling task is the only one that you may delete via this command. There is no return from this call. If you have defined a kill-processing address for this task, RDOS raises it to the highest priority and control returns to the Task Scheduler. Otherwise, control returns to the Task Scheduler so that it can allocate system resources to the ready task of highest priority.

Format

```
.KILL
```

Possible Errors

None.

.AKILL

Kill all tasks of a given priority

This command first raises all tasks of a given priority to the highest priority, and then either kills them or transfers control to their kill-processing addresses. All TCBs that it deletes from the active queue are placed in the free TCB chain. This command also immediately kills any tasks suspended by the .XMTW, .TIDS, .REC, or .SUSP calls. An attempt to kill a task waiting for completion of a system call will not succeed until the system call has executed. If the calling task itself belongs to the specified priority, RDOS deletes it.

Required Input

AC0 - Priority class of the tasks you wish to kill.

Format

```
.AKILL
normal return
```

Possible Errors

None. If no tasks exist with the priority specified in AC0, RDOS takes no action.

.ABORT

Abort a task

This command readies a specified task immediately and instructs it to execute the equivalent of task call .KILL when it gains CPU control. If a kill-processing address exists, RDOS transfers control to it. The exact time of completion depends on the internal priorities of the system. For example, a task attempting to perform a sequential write of 500 bytes might be aborted after writing any number of bytes. You use an ID number to specify the task you want to abort. Thus, the caller can abort either itself or some other ready or suspended task.

Task call .ABORT does not release open channels or overlays used by the aborted task. All outstanding operations performed by the task, such as message transmission or reception, are terminated. Likewise, all *system calls* are aborted, with two exceptions: (1) calls performing multiplexor or MCA I/O, and (2) System read or write operator message calls, such as the .RDOPR and .WROPR described in Chapter 6.

Your program can abort multiplexor or MCA I/O by closing their channel(s). Operator messages initiated by *task calls* .TRDOP and .TWROP can also be aborted. (Only messages initiated by the system call versions, .RDOPR and .WROPR, are not aborted; a single program cannot use both task and system versions of these calls.)

Required Input

AC1 - ID of the task to be aborted.

Format

```
.ABORT
error return
normal return
```

The contents of AC0 are lost upon return.

Possible Errors

AC2	Mnemonic	Meaning
61	ERTID	An ID of zero was specified, or no such task ID was found.
110	ERABT	The specified task was in the process of performing multiplexor or MCA I/O; of performing a system read/write operator message call; or of being aborted by another task.

Task State Modification

This section describes commands that modify the priority or state of a task. In order of discussion, they are:

.PRI	Change the calling task's priority.
.ARDY	Ready all tasks of a given priority.
.SUSP	Suspend the calling task.
.ASUSP	Suspend all tasks of a given priority.

.PRI

Change the calling task's priority

This command changes the priority of the calling task to the value contained in AC0. RDOS assigns this task the lowest priority in its new priority class; the Task Scheduler allocates CPU control to all other ready tasks in the same class before passing control to this one. Naturally, its position in this priority class will change as rescheduling proceeds.

Required Input

AC0 - New priority value for the calling task. If you request a priority higher than 377_8, RDOS accepts only the value in bits 8 through 15.

Format

```
.PRI
normal return
```

Possible Errors

None.

.ARDY

Ready all tasks of a given priority

This command readies all tasks that have been suspended by the .ASUSP, .SUSP, or .TIDS commands and that share the priority you specify in AC0. That is, the .ARDY command resets bit U in word TPRST of each Task Control Block that was set by a previous call to .ASUSP, .SUSP, or .TIDS. Tasks suspended for any other reason (eg, outstanding system calls) will not be readied until bit S of word TPRST is also reset (eg, by receiving a task message via the .REC command). RDOS cannot ready a task until the program environment allows it to zero bits S and U of word TPRST in the task's TCB.

Required Input

AC0 - Priority of task(s) you wish to ready.

Format

```
.ARDY
normal return
```

Possible Error

None. If there are no tasks of the priority given in AC0, RDOS takes no action.

.SUSP

Suspend the calling task

This command suspends the calling task by setting bit U of that task's TCB to one. The task remains suspended until your program readies it with the .ARDY or .TIDR command.

Format

```
.SUSP
normal return
```

Possible Errors

None.

.ASUSP

Suspend all tasks of a given priority

This command suspends all tasks of the priority you specify in AC0. The calling task may suspend itself with this call. All tasks suspended by .ASUSP—even those suspended for other reasons, such as an outstanding system call or setting bit S of TPRST—remain suspended until readied by an .ARDY or .TIDR command.

Required Input

AC0 - Priority of the task(s) you wish to suspend.

Format

```
.ASUSP
normal return
```

Possible Errors

None. If no tasks exist with the priority given in AC0, RDOS takes no action.

Inter-task Communication

RDOS provides a mechanism that allows single tasks to transmit and receive one-word messages. You can also use this mechanism to lock a task process and prevent multiple tasks from entering the process concurrently. Your program specifies an address for the one-word message, and must clear this address to 0 before depositing the message via a transmit call. If several tasks attempt to receive a message from the same address, only the task of highest priority will receive the message.

.XMT and .XMTW

Transmit a message and wait

These commands instruct the calling task to send a one-word, nonzero message to an empty (all zero) message location for another task. If a task has issued call .REC for this location, it will receive the message and be readied. If no .REC command is outstanding, RDOS deposits the message. The .XMTW command does not return until the message has been received, while the .XMT command returns as soon as the transmitting task is readied.

Required Input

AC0 - The address in user address space where you want to deposit the message. This address must not have bit 0 set to 1.

AC1 - The one-word, nonzero message that RDOS will pass to the address in AC0, for the receiving task.

Format

```
.XMT or .XMTW
error return
normal return
```

Possible Errors

AC2	Mnemonic	Meaning
43	ERXMT	The message address is already in use.
115	ERXMZ	Zero message word.

.IXMT

Transmit a message from a user interrupt service routine

This command enables an interrupt routine to send a message to a task in the current environment. The .IXMT command issued from an interrupt routine has the same effect as the .XMT command issued from a task.*

As Chapter 7 explains, your program can specify a user-defined device—that is, a device not defined during system generation—via the .IDEF command. When a user-defined, device interrupt occurs, control passes to the interrupt service routine that you have written for the device. RDOS freezes the entire task environment while the interrupt routine executes; the routine ends with task call .UIEX. If AC1 contains 0 at call .UIEX, RDOS restarts the environment at its former state; if AC1 contains nonzero, it forces rescheduling. If the message sent to a task will affect the environment, you may want to force rescheduling on exit from the interrupt routine.

Even though the task environment may be frozen, RDOS immediately readies a task that has issued a .REC call for the message that it is intended to receive via .IXMT. The contents of all accumulators are destroyed upon return from .IXMT. Hence, your program must restore AC3 and AC2 (if unmapped) before attempting to exit from the service routine via .UIEX. For full details, refer to Chapter 7 under "Servicing User Interrupts."

Required Input

AC0 - Location of the message. The contents of this location must be zero before you invoke the .IXMT command.

AC1 - The nonzero message you want to transmit.

Format

```
.IXMT
error return
normal return
```

Possible Errors

AC2	Mnemonic	Meaning
43	ERXMT	Message address is already in use.
115	ERXMZ	Zero message word.

*The IXMT command and certain other user interrupt calls are not really task calls, since you can issue them only from an interrupt-processing routine. When you use them, the task scheduler and task environment are in suspension. Refer to Chapter 7 for details.

.REC

Receive a message

This command returns a message in AC1 that another task (or interrupt service routine) has posted by a transmit command, and restores the contents of the message address to all zeroes. The message address must be lower than 2_{15}, and bit 0 must not be set.

If a task issues a .REC command and no other task has posted a message to the message address, the receiving task remains suspended until the message is sent. If the message has already been issued and if the receiving task has not also been suspended by an .ASUSP or .TIDS command, control returns to the Task Scheduler. Otherwise, the task remains suspended until you ready it with task call .ARDY. If several tasks attempt to receive the same message, only the task of highest priority will receive it.

Required Input

AC0 - The message address.

Format

```
.REC
normal return
```

Possible Errors

None. RDOS returns AC2 unchanged.

Locking a Process Via Transmit and Receive Commands

You can use the .REC and .XMT commands to lock and unlock a process or database shared by several tasks, and to prevent more than one task at a time from accessing the database or process path. To do this, your program must define a synchronization word, the message location, to which all tasks will issue a .REC comand. The task in control of the locked resource then issues call .XMT to the synchronization word when it wants to open the resource to other, waiting tasks. RDOS then readies the task of highest priority waiting to receive (.REC) the synchronization word, and gives it unique control of the resource. This task, in turn, uses and then unlocks the resource for another task, and so forth.

Your program must initialize the locking facility before any tasks can use it. It can do this by initially setting the synchronization word to a nonzero value, or by having an initialization task issue .XMT to the synchronization word.

User Overlay Management

In a multitask environment, different tasks can compete for an overlay node, or can use the same overlay simultaneously. These factors create the need for overlay management strategies that do not apply in single-task environments. The commands described in this section enable you to handle user overlays effectively in multitask programs. In order of discussion, they are:

.TOVLD	Load a user overlay.
.OVREL	Release an overlay.
.OVEX	Release an overlay and return to the caller.
.OVKIL	Kill the calling task and release its overlay.

The .TOVLD command is the task call version of .OVLOD, and should always be used to load overlays in a multitask program. If you use system call .OVLOD, only one task in the program can load overlays; moreover, the two calls cannot be included in the same program. When the .TOVLD command is used, the maximum number of overlay nodes you can reserve is 125.

As part of its resource management activities, the Task Scheduler maintains a record, called the *overlay use count* (OUC), of the number of tasks using a currently-resident overlay. It keeps the OUC in an overlay directory created by RLDR for each node in your program. (See Appendix E.)

A ready task can request an overlay (via .TOVLD) either by segment and overlay number, or by symbolic name if you assign the name via an .ENTO pseudo-op. Whenever a task requests an overlay, RDOS checks the overlay directory and the overlay request for certain parameters. If the parameters permit, RDOS loads the overlay into the node, increments the OUC by 1, and gives control to the Scheduler. If the parameters disallow the load, RDOS suspends the calling task (bit T of TPRST) and passes control to the Scheduler; the task will be readied and the overlay loaded when the parameters permit. These actions occur each time a task requests an overlay load.

Every time a task releases a resident overlay (via the .OVREL, OVEX, or .OVKIL commands), the overlay's use count is decremented by 1. The overlay currently occupying the node is not released (allowing a task to load another overlay into the node) until the OUC reaches 0. When the use count equals 0, another task can load a new overlay, resulting in an OUC of 1.

An unconditional disk overlay request (not virtual) guarantees a fresh copy of the overlay. A conditional overlay request loads the overlay only if it is not already in memory; if the overlay is memory-resident, RDOS increments the OUC by 1. Conditional loads can save time, but may be used only for reentrant overlays. As mentioned in Chapter 4, it is recommended that all your overlays be reentrant; if any overlay is not, a task requiring its use must load it unconditionally.

.TOVLD

Load a user overlay

This command requests the use of the appropriate overlay node and the loading of the overlay whose node and number you specify in AC0.

If you did not assign a symbolic name to the overlay via .ENTO before loading the program, you must pass the node number that it will occupy in the left byte, and its overlay number in the right byte, of AC0. The node number corresponds to the segment number within the overlay file. The first segment, number 0, is defined by the first set of brackets in your RLDR command line; it corresponds to node 0 in memory.

The overlay number is the relative position of the overlay within its segment. Segment 0's overlays are numbered 0, 1, and upward sequentially through n. The second segment loaded is segment 1, corresponding to node 1; its overlays are also numbered sequentially from 0 through n, and so forth.

You can specify either a conditional or unconditional load in AC1. If the load request is conditional and the node is free, RDOS loads the overlay. If the node already contains the requested overlay, RDOS returns to the Scheduler immediately. Because another task is also using the overlay, it must be reentrant. If another overlay currently occupies the node and its OUC is a nonzero value, the caller is suspended until the node becomes free.

If the load request is unconditional and the node is free, RDOS loads the overlay whether it is currently memory-resident or not. If the overlay use count has not decremented to zero (freeing the node), the caller is suspended (bit T of TPRST) until the node becomes free. Figure 5.3 charts the sequence that RDOS follows when you issue the .TOVLD command.

Required Input

AC0 - Overlay node/number word.

AC1 - For a conditional load, pass 0. For an unconditional load, pass -1.

AC2 - The channel number on which you opened the overlay file. (See the description of .OVOPN in Chapter 4.

Format

```
.TOVLD
error return
normal return
```

Note that you must pair all overlay load requests with an eventual overlay release (.OVREL/.OVKIL) or the node will be reserved indefinitely. Also note that under certain conditions—such as a nonmatching save and overlay file—the left byte of AC2 may be nonzero on an error return.

Possible Errors

AC2	Mnemonic	Meaning
37	EROVN	Invalid (nonexistent) overlay name or segment.
40	EROVA	Overlay file is not a contiguous file.
101	ERDTO	Ten-second disk timeout occurred.

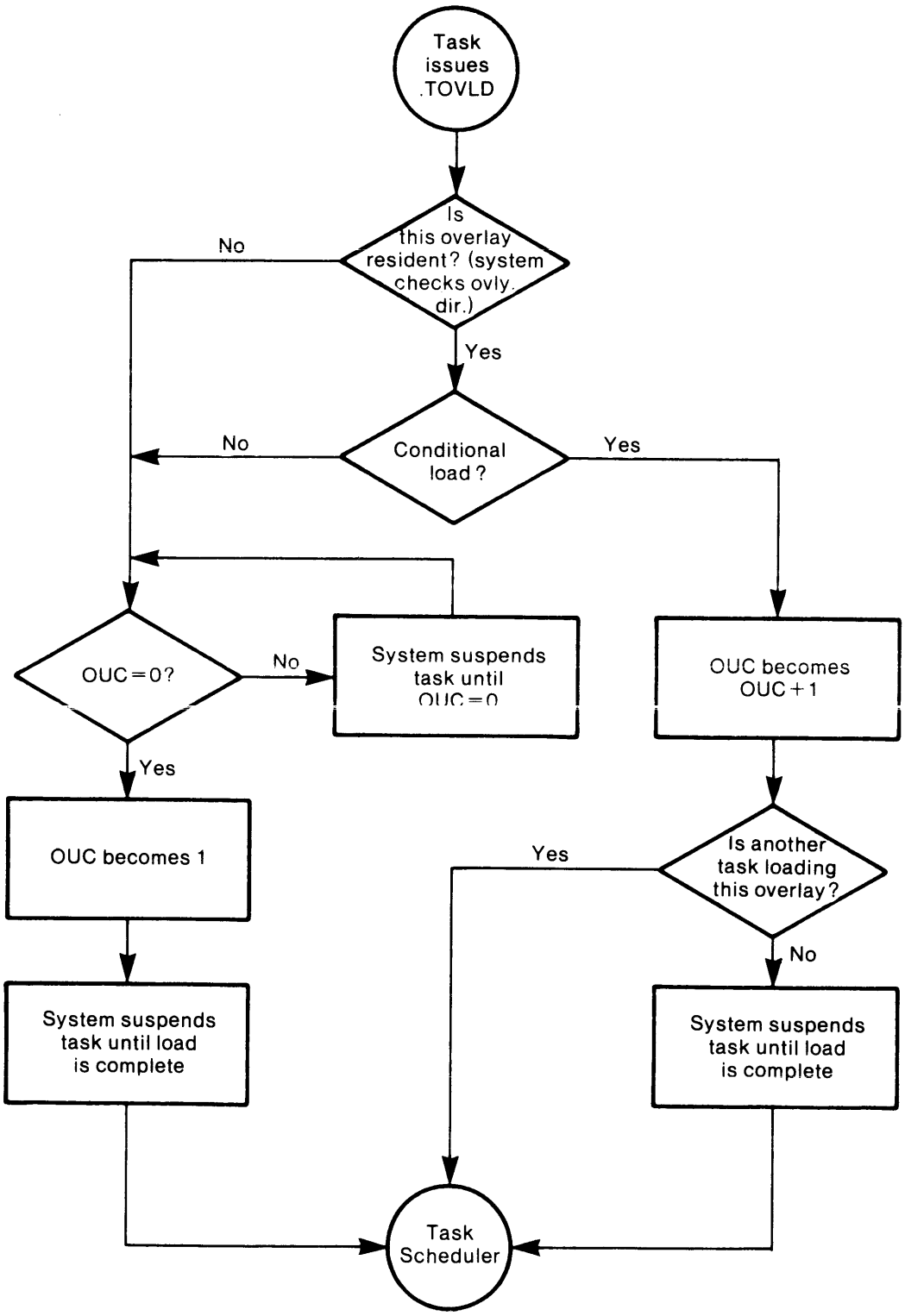


Figure 5.3 TOVLD logic sequence

SD-00540

.OVREL

Release an overlay

This command decrements the overlay use count (OUC) and releases the node if the use count equals zero. The overlay that you wish to release must not issue this command.

Required Input

AC0 - Overlay node/number word. Pass the node number in the left byte and the overlay number in the right byte.

Format

```
.OVREL
error return
normal return
```

Possible Errors

Only one possible error results from this command. Its mnemonic is EROVN, signifying an invalid overlay node/number, or that the overlay node is not occupied by the overlay specified. RDOS passes error code 37 in AC2 when this error occurs.

.OVEX

Release an overlay and return to the caller

This command decrements the overlay use count (OUC) and releases the node if the use count equals 0. Additionally, control returns to an address specified by the caller—typically the return address of the caller if returning from a subroutine within an overlay.

Required Input

AC0 - Overlay node/number word.

AC2 - Return address upon successful execution of this call.

Format

```
.OVEX
error return
```

Possible Errors

Only one possible error message results from this command. Its mnemonic is EROVN, signifying an invalid overlay number, or that the overlay node is not occupied by the overlay specified. RDOS passes code 37 in AC2 when this error occurs.

.OVKIL

Kill the calling task and release its overlay

This command kills the caller and decrements the overlay use count; it also releases the node if the OUC equals 0. This is the conventional method of terminating a queued, overlayed task. The overlay that you wish to release can issue this call.

Required Input

AC0 - Overlay node number in the left byte; overlay number in the right byte.

Format

```
.OVKIL
error return
```

Possible Errors

Only one possible error results from this command. Its mnemonic is EROVN, signifying an invalid overlay number, and RDOS returns error code 37 in AC2 when it occurs.

Enqueuing Tasks

.QTSK

Queue a memory-resident or overlay task

This command periodically initiates a task and queues it for execution. If the task resides within an overlay, this command loads the overlay. You need not issue .TOVLD for an overlayed task, but the .QTSK mechanism requires that you declare .TOVLD external, via an .EXTN statement, in the program. If no TCB is currently available for the creation of the new task, RDOS executes this command as soon as a TCB becomes available. If two tasks are queued for execution at the same time of day, the task of highest priority receives control first. (Appendix C demonstrates the use of .QTSK and overlays in a real-time programming example.)

A task created and queued by .QTSK resembles any other task, and it is your responsibility to kill or suspend it after it has performed its function. If it resides in an overlay, it can kill itself and release the overlay node via the .OVKIL command. (If the overlay node is not released, no other task will be able to use it.)

If RDOS does not take the error return, control returns to the task issuing the call at the normal return based on the task's priority; the calling task is not suspended. When the queued task gains control, AC2 contains a pointer to the Task Queue Table.

If your program does not declare either .TOVLD, OVKIL, OVREL, or .OVEX external via an .EXTN statement, RDOS executes the equivalent of an .ERTN command, and passes error code 117 (ERQOV) in AC2.

The .QTSK command needs no input to AC0 or AC1, but requires you to build a table of specifications for the new task and to input the starting address of this table in AC2. The table must be QTLN* words long and contain the entries shown in Table 5.3.

*These symbols are defined under USER TASK QUEUE TABLE in the listing of file PARU.SR in Appendix B.

Displacement	Mnemonic	Meaning
0	QPC	Starting address of task
1	QNUM	Number of times to queue the task (-1 if task is to be queued an unlimited number of times)
2	QTOV	Symbolic name or node number/overlay number (-1 for a memory-resident task)
3	QSH	Starting hour (-1 if task is to be queued immediately)
4	QSMS	Starting second in hour (reserved but unused if QSH = -1)
5	QPRI	Task ID/task priority
6	QRR	Rerun time increment in seconds
7	QTLNK	System word
10	QOCH	Overlay channel (unused by memory-resident tasks)
11	QCOND	Conditional/unconditional load flag (unused by core-resident tasks)

Table 5.3 User task queue table

According to Table 5.3, entry OPC must contain the entry address in the overlay or memory-resident task where control will be directed when RDOS raises the task to the executing state.

Entry QNUM is an integer value describing the number of times the task will be queued. The task is queued QNUM times—or without limit if QNUM equals -1—unless you issue task call .DQTSK. This call halts the queuing of the specified task. RDOS decrements QNUM each time it queues the task.

Entry QTOV must contain the overlay's .ENTO name or its number in the left byte, and the overlay number in the right byte for overlay tasks; for memory-resident tasks, set this word to -1. If you did not assign a symbolic name to the overlay via .ENTO, you must use the segment/node and overlay numbers assigned by the loader. Make sure that the values of QTOV correspond to the values assigned at load time.

Entries QSH, QSMS, and QRR all affect the time at which RDOS creates the task. QSH sets the hour to execute, and QSMS sets the second within that hour that the task will be created. If QSH contains -1, RDOS creates the task immediately; if QSH occurs before the current time of day, or is greater than 24 but less than 48 hours, RDOS queues the task for the next day; and if QSH equals (24*d) + h, RDOS queues the task in d days. Entry QRR sets the interval (in seconds) between the times the task will be queued.

Entry QPRI contains the task ID (if any) in its left byte and the task priority in its right byte. If a task with the same ID exists at the time that RDOS activates the task, the system clears this task's ID number to zero.

The system maintains word QTLNK.

Entry QOCH must contain the number of the channel on which you opened the overlay file with an .OVOPN command. Entry QCOND must contain -1 if you want the overlay load to be unconditional. Both entries are unused by memory-resident, queued tasks.

QAC2 is used as a temporary storage area by RDOS.

Required Input

AC2 - Pointer to the task queue table.

Format

```
.QTSK
error return
normal return
```

On the normal return, AC2 contains the contents of .QAC2.

Possible Errors

AC2	Mnemonic	Meaning
50	ERQTS	Illegal information in Task Queue Table.
117	RQOV	.TOVLD not loaded for an overlay queued task.

.QTSK Example

To demonstrate the use of the .QTSK command, Figure 5.4 shows its application in a closed-circuit, television display network of airline arrivals and departures. The figure contains excerpts from a main program in which one overlayed task checks a central control panel for each arrival and departure, and displays it, along with pending or recent arrivals and departures, on network screens throughout the terminal. The amount of air traffic varies with the time of day; accordingly, .QTSK adjusts the interval at which the task checks the control panel. Figure 5.4 shows .QTSK code for 12:30 p.m., a time of relatively slow traffic; thus, .QTSK specifies a 60-second check on the panel.

```
.
.
.EXTN           .QTSK           .TOVLD  .DQTSK  .OVKIL  etc.      ;DECLARE ALL RELEVANT
                                                                   ;CALLS EXTERNAL.
.
.
.
LDA 2,          .TABLE
.QTSK
.
.
.
TABLE           :TABLE
TABLE           :START          ;STARTING ADDRESS OF PANEL MONITOR TASK.
                                ;QUEUE THE TASK CONTINUOUSLY (UNTIL
                                ;A DQTSK AND NEW QTSK CHANGE THE INTERVAL).
01214                           ;GET THE TASK FROM OVERLAY 01214
                                ;IN THE OVERLAY FILE.
12.                             ;QUEUE THE TASK AT THE 12TH HOUR.
30.*60.                         ;30 MINUTES PAST THE HOUR.
7*400+4                         ;THE TASK'S ID IS 7, AND ITS PRIORITY
                                ;WILL BE 4.
60.                             ;QUEUE THE TASK FOR EXECUTION
                                ;EVERY 60 SECONDS.
0                               ;RDOS WILL USE THIS WORD.
3                               ;THE PROGRAM'S OVERLAY FILE
                                ;WAS OPENED ON CHANNEL 3.
-1                              ;LOAD THE OVERLAY UNCONDITIONALLY.
0                               ;RDOS WILL USE THIS WORD,
0                               ;AND THIS WORD.

.
.
```

Figure 5.4 QTSK example

.DQTSK

Dequeue a memory-resident or overlay task

This command dequeues a task which has been queued for execution by task call .QTSK. In effect, the .DQTSK command bypasses the value currently stored in QNUM of the queued task's queue table. (See Table 5.3.) If, at some later moment, the task is requeued by a call to .QTSK, the queuing process resumes its normal course since .DQTSK does not actually modify the contents of QNUM.

Required Input

AC1 - ID of the task to be queued.

Format

```
.DQTSK
error return
normal return
```

Upon a normal return, AC2 returns the base address of the task's queue table.

Possible Errors

Only one possible error results from this command. Its mnemonic is ERTID, signifying a task ID error, and RDOS returns code 61 in AC2 when it occurs.

User/System Clock Commands

All system clock commands can be issued from either a single-task or multitask environment. These commands are of little practical use in a single-task environment, however, and are presented here for that reason. The commands in this section permit your program to define, exit from, and remove a clock driven by the system's Real Time Clock (RTC). In order of discussion, they include:

.DELAY	Delay the calling task's execution.
.DUCLK	Define a user clock.
.UCEX	Exit from a user clock routine.
.GHRZ	Examine the system's RTC frequency.

The Real Time Clock suspends the environment at the intervals you define, and passes control to the routine whose address you specify. You can exit from this routine and return to the environment via system call .UCEX. You may not issue any system or task calls (other than .IXMT, .SMSK, or .UCEX) from this routine because RDOS freezes all multitask activity, just as it does for a user interrupt. (See Chapter 7.) Any user clock routine executes in the interrupt world, not in program space; for this reason, you should make sure that your routine is correct.

.DELAY

Delay execution of the calling task

This command suspends the calling task for the number of real time pulses indicated by AC1. You set the Real Time Clock's frequency during system generation. (See and check it via the .GHRZ command described later in this section.)

The accuracy of the .DELAY command can be affected by three variables:

- The frequency of the Real Time Clock, as set during system generation
- The priority of the issuing task, compared to other tasks
- The priority of the issuing program (ground) compared to the other program.

RTC pulses are not synchronized with the .DELAY call; thus, it may be unrealistic to request single-pulse delays. Single-pulse delay requests can be delayed anywhere between 0 and 1 RTC pulse.

Required Input

AC1 - Number of RTC pulses.

Format

```
.SYSTM
.DELAY
error return
normal return
```

Possible Errors

None. The error return is never taken. You lose the contents of AC1 upon return.

.DUCLK

Define a user clock

This command defines a user clock, which will be entered at the intervals you specify in AC0. When this interval expires, RDOS suspends the Task Scheduler and multitask environment, if any; control then goes to the address specified in AC1. Each time control passes to this address, AC0 contains a value indicating where control came from at the interrupt. AC0 contains -1 if control originated from the system while it was in an idle loop (ie, awaiting an interrupt); it contains 100000 if the other ground's program held control; or it contains the current PC if control originated from your program.

When control passes to your user clock routine, AC3 contains the address of the return upon entry to the user routine. In unmapped systems, you must use this address in the .UCEX command to return to the multitask environment.

Required Input

AC0 - The integer number of system RTC cycles that you want to elapse between each clock interrupt.

AC1 - The address of the routine to receive control when each interval expires. Note that no system or task calls (excepting .UCEX, .IXMT, or .SMSK) can be issued from this routine. Nor should assembly instruction INTEN be issued in an unmapped system.

Format

```
.SYSTM
.DUCLK
error return
normal return
```

Possible Errors

AC2	Mnemonic	Meaning
45	ERIBS	A user clock already exists.
74	ERMPR	Mapped systems only: address outside address space.

.UCEX

Exit from a user clock routine

When RDOS enters a user clock interrupt routine, it places the return address in AC3. In an unmapped system, RDOS requires this address to return to the multitask environment; thus, if your interrupt routine uses AC3, it must restore this accumulator before issuing the .UCEX command. In a mapped system, RDOS ignores the value input in AC3 when you issue this command. In all systems, RDOS reschedules both the task and program environments only if AC1 contains a nonzero value upon exit. Control returns to the point where the .DUCLK interrupt occurred. You may issue this command in a single-task environment.

Required Input

AC1 - Zero to continue the environment; nonzero to reschedule.

AC3 - Return address to routine (unmapped systems only).

Format

```
.UCEX
```

Possible Errors

None.

.RUCLK

Remove a user clock

This system call removes a previously defined user clock from the system.

Format

```
.SYSTM
.RUCLK
error return
normal return
```

Possible Errors

Only one possible error results from this command. Its mnemonic is ERIBS, indicating that no user clock is defined, and RDOS passes code 45 in AC2 when it occurs.

.GHRZ

Examine the system real time clock

This system call returns a code for the Real Time Clock frequency in AC0. The possible codes and their meanings are:

0 There is no Real Time Clock in the system.

1 Frequency is 10 HZ.

2 Frequency is 100 HZ.

3 Frequency is 1000 HZ.

4 Line Frequency is 60 HZ.

5 Line Frequency is 50 HZ.

Format

```
.SYSTM
.GHRZ
error return
normal return
```

Required Input

None.

Possible Errors

None.

Managing Tasks by ID Number

This section describes commands that manage tasks according to the ID number specified in AC1. In order of discussion, they include:

.IDST	Get a task's status.
.TIDP	Change a tasks's priority.
.TIDR	Ready a task.
.TIDS	Suspend a task.
.TIDK	Kill a task.

.IDST

Get a task's status

This command returns a code in AC0 describing a task's status. The possible codes and their meanings are:

0 Ready

1 Suspended by a .SYSTM call or .TRDOP command

2 Suspended by a .SUSP, .ASUSP, or TIDS command

3 Suspended by a .XMTW or .REC command

4 Waiting for an overlay node

5 Doubly suspended by .ASUSP, .SUSP, or .TIDS and by .SYSTM

6 Doubly suspended by .XMTW or .REC and .SUSP, .ASUSP, or .TIDS

7 Waiting for an overlay node and suspended by .ASUSP, .SUSP, or .TIDS

10 No task exists with this ID number

Required Input

AC1 - The task's identification number.

Format

```
.IDST
normal return
```

On the normal return, RDOS passes a status code in AC0 and the base address (displacement TCB) of the task's TCB in AC2.

Possible Errors

None.

.TIDP

Change a task's priority

This command changes the priority of the task whose ID you specify in AC1.

Required Input

AC0 - The new priority (from 0 to 255 inclusive) in the right byte (bits 8 through 15).

AC1 - ID of task.

Format

```
.TIDP
error return
normal return
```

Possible Errors

Only one possible error results from this command. Its mnemonic is ERTID, indicating an erroneous task ID, and RDOS returns code 61 in AC2 when it occurs.

.TIDR

Ready a task by ID number

This command readies only the task whose identification number you place in AC1. It resets bit U in word TPRST of this task's TCB, which was set by a previous call to .ASUSP, .SUSP, or .TIDS. If bit U has already been reset, RDOS takes the normal return.

Required Input

AC1 - ID number of the task you wish to ready.

Format

```
.TIDR
error return
normal return
```

Possible Errors

Only one possible error results from this commend. Its mnemonic is ERTID, indicating an erroneous task ID, and RDOS returns code 61 in AC2 when it occurs.

.TIDS

Suspend a task by ID number

This command suspends only the task whose identification number you pass in AC1. It sets bit U in word TPRST of the specified task's TCB. If bit U in word TPRST is already set, RDOS takes the normal return.

Required Input

AC1 - ID number of the task you wish to suspend.

Format

```
.TIDS
error return
normal return
```

Possible Errors

Only one possible error results from this command. Its mnemonic is ERTID, indicating that no task exists with the specified ID number, and RDOS returns code 61 in AC2 when it occurs.

.TIDK

Kill a task by ID number

This command kills only the task whose identification number is specified in AC1. RDOS raises the task to the highest priority (0); places it at the end of that priority chain; and transfers it to a kill-processing address (if any) or terminates it. If the task is executing a system call, it will not be killed until the system call is completed.

Required Input

AC1 - ID number of the task you wish to kill.

Format

```
.TIDK
error return
normal return
```

Possible Errors

Only one possible error results from this command. Its mnemonic, ERTID, signifies a task ID error, and RDOS returns code 61 in AC2 when it occurs.

Task/Operator Communications Calls

This section describes two commands, .TWROP and .TRDOP, that a task can issue to communicate with the system console, $TTO/$TTI. You can use these calls to interact directly with tasks in your program via OPCOM commands, discussed in the next section; or you can use the task calls or OPCOM commands alone. To use either (or both) features, you must have selected the option of operator messages during system generation. If your program uses operator message calls or OPCOM commands, you must specify an extra task in the RLDR command line to provide a TCB for system use. The format of console commands is similar for the task calls and OPCOM messages.

Note that your program cannot use both system and task versions of the operator message calls. The system versions, .WROPR and.RDOPR, are described in Chapter 6.

.TWROP

Write a task message to the console

This command instructs the calling task to write an ASCII string to the system console, $TTO. The message may include up to 129 characters, including the required carriage return, form feed, or null terminator. RDOS always displays two exclamation points (!!), along with the letters "B" or "F," before it displays the text string. These letters indicate that a background (B) or foreground (F) task issued the message. Then, depending on your input, RDOS displays the task's ID number and the message. Thus, the format of task messages to the console is:

!!F [TID] message or *!!B [TID] message*

If AC1 contains -1 when the task issues call .TWROP, RDOS displays the three-character prefix (!!F or !!B) followed by a message string of up to 129 characters, including the required terminator. If AC1 contains a value other than -1 on this call, the first four characters of the message area are overwritten by the three octal digits of the task ID number and one space. Text written to the console includes a three-character prefix (!!B or !!F) followed by the task ID number and the remainder of the message—a string of up to 124 characters, including the terminator.

More than one task may have an outstanding request to write task messages to the console. However, the save file cannot include both task and system calls to read or write messages to or from the console. Several tasks can use the same message string (same byte pointer), but only if you suppress TID information. Note that the .TWROP command requires an extra TCB in the program.

Required Input

AC0 - Byte pointer to area that holds the message. (If AC1 does not equal -1, this area must include a four-byte null prefix to receive the task ID and space separator.)

AC1 - Specify -1 to suppress the task ID, or some other value to display the ID (see AC0, above).

Format

```
.TWROP
error return
normal return
```

Possible Errors

AC2	Mnemonic	Meaning
74	ERMPR	Address outside address space.
120	EROPM	Operator messages not specified during system generation.

.TRDOP

Read a task message from the console

This task call prepares the calling task to receive a message from the system console, $TTI. The task issuing this call may reside in either the foreground or background program areas, and more than one task may issue an outstanding request for a task message. However, if you use task calls .TWROP/.TRDOP to write or read messages to or from the console, you cannot also use system calls .WROP/.RDOP within the save file.

You must type CTRL-E as the first character (echoed on the console as an exclamation point, !). If the cursor is not at column 0, press the RETURN key first. The second character must be either an F or B to indicate whether the task resides in the foreground or background. If you type some character other than F or B in column 2, RDOS sounds the console bell as a warning and accepts no further characters until you provide the correct input.

After letters F or B, type the ID of the task to receive the message, followed by a comma delimiter; then type the message itself immediately after the comma. The last character in the message string must be a carriage return, form feed, or null terminator. Your input—including CTRL-E, letters B or F, the task's ID and comma, your message, and the terminator—should not exceed 132 characters. The required format for an input message is as follows, where angle brackets indicate an ASCII character:

⟨ CTRL E ⟩ F or TID, message ⟨CR⟩

If, after pressing CTRL-E, you want to cancel the message transmission, press the RUBOUT key. This key erases a command or message starting with the most recent character typed. On TTY consoles, a left arrow ← is echoed for each rubout.

Remember to specify an extra TCB for the .TRDOP command in your RLDR command line (or two extra TCBs for both reads and writes). There must also be one TCB available for use by the system. RDOS uses this TCB to create a task to monitor the $TTI keyboard for task—keyboard messages, allowing one or more tasks to issue the .TRDOP command.

RDOS can dislay two messages to indicate errors in messages intended for tasks. These messages and their meanings are:

TID NOT FOUND	No task with the specified ID number was waiting for a console message.
INPUT ERROR	Nonnumeric character found in task ID.

Required Input

AC0 - Byte pointer to message area. RDOS will not transmit the task ID and comma to the message area.

Format

```
.TRDOP
error return
normal return
```

On the normal return, RDOS gives the byte count in AC1 (including the terminator but excluding the task ID and delimiter).

Possible Errors

AC2	Mnemonic	Meaning
42	ERNOT	Out of TCBs (ie, there is no TCB available to monitor the console).
74	ERMPR	Mapped systems only: address outside address space.
120	EROPM	Operator messages not specified during system generation.

Task/Operator Communications Module (OPCOM)

This section presents the task/operator communications package, OPCOM, which allows you to use console commands to check or change the status of tasks, and to run these tasks or queue them for periodic execution.

OPCOM is unrelated to the Command Line Interpreter (CLI), and has its own syntax and command definitions. OPCOM has a limited command repertoire since it—unlike the CLI—is part of the save file with which it is being used.

Following a discussion of command line syntax, the .IOPC command, which initializes the OPCOM package, is introduced. Then the OPCOM commands themselves are described. In order of discussion, they are:

DEQ	Dequeue a queued task.
KIL	Kill a task.
PRI	Change a task's priority.
QUE	Queue a task for periodic execution.
RDY	Ready a task.
RUN	Execute a task.
SUS	Suspend a task.
TST	Display a task's status.

An example at the end of the section demonstrates the use of these commands.

In addition to the OPCOM module, this package requires modules OPMSG (unmapped) or MOPMS (mapped). The RLDR program loads these modules if you declare the initialization command, .IOPC, external via an .EXTN statement. You must also specify an extra TCB (or two for reads and writes) for RDOS, unless you have included one or two extra TCBs for the operator task calls described earlier. In addition, you must have selected the option of operator messages during system generation.

The OPCOM module requires approximately 457_8 NREL words, while the OPMSG or MOPMS modules require approximately 472_8. Thus, you will need a total of roughly 1150_8 NREL words for any system.

Each OPCOM command evokes a task call that performs the desired function; accordingly, you will find details on the internal operation of each command under its related task call (eg. QUE and .QTSK). Each OPCOM command requires that you enter a program number for the task; you specify this number in a table that you build for each task before initializing OPCOM. The program number can be the task ID number, or not. Certain commands require ID numbers, while others (RUN and QUE) require program numbers; to avoid confusion, it is recommended that you use the task's ID number as its program number. After initializing the communications package, you can enter commands using the format and syntax described next. OPCOM responds with the message OK if it has executed a command, or with one of four descriptive error messages.

OPCOM Command Syntax

OPCOM has been designed to accept a limited number of keyboard commands to keep the command processor small, since it must always remain a resident part of the save file. All OPCOM commands have the following, fixed format, where angle brackets indicate an ASCII character and brackets surround any optional input:

⟨ CTRL-E ⟩ B or F *,command,task,[arg1, ... argn]⟨CR⟩

You enter CTRL-E by pressing the CTRL and E keys simultaneously. If the cursor is not at column 0, press the RETURN key first. You must then type either B or F to indicate whether the save file being commanded is in the background or foreground. Both background and foreground programs use $TTI/$TTO. Immediately following letters B or F, type an asterisk followed by a comma. Enter the OPCOM command immediately after the comma. Follow the command with a comma and one or more task arguments; separate multiple arguments by commas. Terminate the command line with a carriage return.

Note that the command structure is rigid; if you depart from the command format (eg, use spaces or delimiters), OPCOM rejects the command and displayss the messge

INPUT ERROR

on the console. When OPCOM has executed a command, it reports to your console as follows:

!!B or *F OK*

.IOPC

Initializing the operator communications package

This command initializes the OPCOM package, and must be issued before you can execute any of OPCOM's commands. If you do not plan to use OPCOM commands RUN and QUE or DEQ, the first three accumulators (AC0, AC1, and AC2) must each contain 0 when you issue the .IOPC call. Otherwise, you must input three parameters to the .IOPC command.

The first of these parameters, passed in AC0, is the address of the queue area reserved for this call. OPCOM needs one queue area frame for each RUN or QUE command. (The QUE command awaits execution until the task has been queued for the last time.) The total queue area is *n**QTLN words long, where *n* equals the number of queue frames and QTLN is the queue frame size. (QTLN is defined in the listinng of PARU.SR found in Appendix B.) The queue area is managed exclusively by OPCOM.

You pass the second parameter in AC1. The left byte must contain the channel number on which you opend the overlay file: if no overlay is involved, this byte must contain 0. The right byte of AC1 must describe the maximum number of different tasks that you will queue or run simultaneously. (This value defines the queue area when multiplied by QTLN.) OPCOM can load overlay tasks on request, but your program must release each node used for these tasks by issuing the .OVKIL or .OVEX commands.

The last parameter, passed in AC2, is the base address (displacement 0) of the task table. This table consists of a series of five-word frames that describe each task to be run or queued. To build this table, use the following specifications:

Displacement	Contents
0	Program number
1	Overlay symbolic name, or node (left byte)/ number (right byte); (-1 if a core-resident task)
2	-1 only if unconditional loading is required
3	Task ID (left byte); task priority (right byte)
4	Task starting address

The program number is distinct from the task ID, but you may assign the same value to them if desired. You can modify the task priority by an appropriate OPCOM command. Terminate the task table series with a word containing -1.

Required Input

Pass the following parameters only if you plan to issue OPCOM commands RUN or QUE. Otherwise, clear AC0, AC1, and AC2 to zero when you issue the .IOPC command.

AC0 - Queue area address.

AC1 - Left byte: overlay channel (or zero). Right byte: maximum number of queues.
AC2 - Task table address.

Format

```
.IOPC
error return
normal return
```

Possible Errors

AC2	Mnemonic	Meaning
42	ERNOT	Out of TCBs.
120	EROPM	Operator messages not specified during system generation.

DEQ

Dequeue a previously queued task

This OPCOM command dequeues the previously-queued task whose ID you specify as an argument. The task argument must be an octal integer ranging from 1 to 377; it cannot be 0. After executing the command, OPCOM displays the message OK; you can then issue another command. If OPCOM cannot execute the command, it displays one of two error messages and await another command.

Format

⟨ CTRL-E ⟩ F or B *,DEQ,task ID ⟨CR⟩

Possible Errors

Message	Meaning
INPUT ERROR	Command syntax error.
TID NOT ACTIVE	No task with the specified task identification number was found.

KIL

Kill a task

This OPCOM command immediately kills the task whose ID you specify as an argument. The task ID argument must be an octal integer in the range of 1 to 377; it cannot be 0. After executing the command, OPCOM displays the message OK; you can then issue another command.

Format

⟨ CTRL E ⟩ F or B *,KIL,task ID ⟨CR⟩

Possible Errors

Message	Meaning
INPUT ERROR	Command syntax error.
TID NOT ACTIVE	No task with the specified task identification number was found.

PRI

Change a task's priority

The PRI command changes the specified task's priority to the priority given as an argument. The task ID and new priority arguments must each be an octal integer within the range 1 to 377. After executing the command, OPCOM displays the message OK on the system console; you can then issue another command.

Format

⟨ CTRL-E ⟩ F or B *,PRI,task ID, new priority ⟨CR⟩

Possible Errors

Message	Meaning
INPUT ERROR	New priority exceeded 377_8, or syntax error detected.
TID NOT ACTIVE	No task with the specified task identification number was found.

QUE

Queue a task for periodic execution

This command creates and periodically executes a task for execution using task call .QTSK's logic. The task may be either memory-resident or an overlay.

If the task resides in an overlay, the QUE command loads the overlay. If no TCB is currently available for the creation of a new task, RDOS waits for one and then carries out this command. If two or more tasks are queued for execution at the same time of day, the task of highest priority receives control first. After this call creates and activates a new task, you must ensure that the system kills or suspends it. If the task resides within an overlay, your program must release the node after the task has executed; otherwise, no other task will be able to use the node.

After successful completion of the QUE command, OPCOM displays the message OK on the system console; you can then issue another command. If OPCOM cannot execute the command, it displays one of four error messages and awaits another command.

Format

⟨ CTRL-E ⟩ F or B *,QUE,program#, ↑⟨CR⟩
[hour,minute,second,repeats,] interval [,priority]⟨CR⟩

In this command line, program # is the number that you chose when initializing OPCOM via the .IOPC command. This argument may be the same as the task ID, or not. Note the use of the up-arrow (↑) to continue a long statement on the next line.

All bracketed entries are optional. If hour is less than the current time of day, or is between 24 and 48, RDOS queues the task for the new day. If hour equals (24*d) + h, RDOS queues the task in d days. To queue for midnight, queue for hour 24. To queue the task immediately, omit hour, minute, and second (but retain their comma delimiters in the command line.

Argument repeats defines the number of times the task will be executed, and interval determines the number of seconds to elapse between each time RDOS queues the task. The interval may not exceed 65,535 seconds (about 18 hours). If argument repeats is omitted, the task is queued an unlimited number of times. (Even if you omit this argument, you must include its comma delimiter.)

The priority argument indicates the priority of the task you want to queue; it is optional because the task's priority (along with other task information) is required in the task table that you input to the .IOPC command. If you enter the priority argument here, it overrides the one you specified to .IOPC. The priority argument is an octal integer; all others are decimal.

Possible Errors

Message	Meaning
INPUT ERROR	One or more required arguments are missing in the command string, or you specified an invalid priority argument.
PROG NOT FOUND	You did not issue the .IOPC call, or did not define the program number in this call (ie, the program table is incomplete).
NO QUEUE AREA	You defined an insufficient number of queue area frames in the call to .IOPC, hence no free queue area is available.
ILLOGICAL QUEUE	You input illegal information in the argument string (RDOS detected this when it passed to .QTSK).

RDY

Ready a task

This command readies the task whose ID you specify as an argument. The task ID must be an octal integer ranging from 1 to 377. After executing this command, OPCOM displays the message OK on the system console; you can then issue another command.

Format

⟨ CTRL-E ⟩ F or B *,RDY, task ID!& ⟨CR⟩

Possible Errors

Message	Meaning
INPUT ERROR	Command syntax error.
TID NOT ACTIVE	No task with the specified task identification number was found.

RUN

Execute a task

This OPCOM command initiates either a memory-resident task or one within an overlay, and queues this task for immediate execution. If the task resides within an overlay, this command loads the overlay. If no TCB is currently available for the creation of a new task, RDOS carries out the command as soon as a TCB becomes free. After creating and activating a task with the RUN command, you must ensure that it is killed or suspended. If the task resides within an overlay, you must release the overlay node.

After completing this command, OPCOM displays the message OK on the system console, indicating that it is ready to accept another command.

Format

⟨ CTRL-E ⟩ F or B *RUN.program #,[priority] ⟨CR⟩

In this command line, program # is the number assigned to this program when you issued the initialization command, .IOPC. This argument may or may not be the same as the task ID, and must be expressed as a decimal integer.

The priority is an optional argument indicating the priority of the task you wish to queue. If you enter the priority here, it overrides the one specified to the .IOPC command when you initialized the OPCOM package. The priority must be an octal integer.

Possible Errors

Message	Meaning
INPUT ERROR	You did not specify a program number in the command line, or you specified an invalid priority.
PROG NOT FOUND	You did not issue the .IOPC call, or did not define the program number in this .IOPC call (ie, the program table is incomplete).
NO QUEUE AREA	You defined an insufficient number of queue area frames in the call to .IOPC; thus no free queue area is available.

SUS

Suspend a task

This command suspends the task whose ID you specify as an argument. The task ID must be an octal integer in the range of 1 to 377. After executing this command, OPCOM displays the message OK on the system console; you can then issue another command.

Format

⟨ CTRL E ⟩ F or B *,SUS,task ID ⟨CR⟩

Possible Errors

Message	Meaning
INPUT ERROR	Command syntax error.
TID NOT ACTIVE	No task with the specified task identification number was found.

TST

Display a task's status

This OPCOM command displays a specified task's status on the console. After executing the command, OPCOM displays the following status on the console:

STAT = s, PRI = ppp

Where s is an octal integer from 0 to 7 representing one of eight states; and ppp is an octal number, ranging from 0 to 377, indicating the task's priority. The possible values of s and their meanings are:

0 Ready

1 Suspended by a .SYSTM call or TRDOP command

2 Suspended by a .SUSP, .ASUSP, or TIDS (SUS) command

3 Suspended by a .XMTW or .REC command

4 Waiting for an overlay node

5 Doubly suspended by .ASUSP, .SUSP, or .TIDS (SUS) and by a .SYSTM call

6 Doubly suspended by .XMTW or .REC and by a .SUSP, .ASUSP, or .TIDS (SUS) command

7 Waiting for an overlay node and suspended by an .ASUSP, .SUSP, or .TIDS (SUS) command

Format

⟨ CTRL E ⟩ F or B *,TST, task ID ⟨CR⟩

The task ID argument must be an octal integer ranging from 1 to 377.

Possible Errors

Message	Meaning
INPUT ERROR	Command syntax error.
TID NOT ACTIVE	No task with the specified task identification number was found.

OPCOM Command Example

Figure 5.5 demonstrates the use of the OPCOM commands discussed in this section. It shows a typical series of console commands and messages, along with explanations of each sequence.

Dialogue	Meaning
!B*,RUN,1↓ *!!B OK* !B*,RUN2↓ *!!B OK*	Run task with program number 1 in the background, and OPCOM verifies execution of the command. Similarly, program 2 is run and is verified.
!B*RUN,3↓ *!!B INPUT ERROR*	An attempt is made to run 3, but OPCOM detects a syntax error (missing comma).
!B*,TST,1↓	Operator requests status of program 1.
!!B STAT=1,PRI=002 !B*,SUS,1↓ *!!B OK*	OPCOM responds with status 1, *ready*, and priority 2. Operator suspends 1 and OPCOM verifies execution of the command.
!B*,TST,1↓	Operator gets status of 1 again;
!!BSTAT=2,PRI=002	status is *suspended by SUS*.
!B*,KIL,1↓ *!!B OK*	Operator Kills program 1, and after system verifies this, operator tries to test its status.
!B*,TST,1↓	
!!B TID NOT ACTIVE	OPCOM responds with error message.

Figure 5.5 Sample console commands and messages DG-25445

Disabling and Enabling the Multitask Environment

This section describes the .SINGL and .MULTI commands, which disable and enable the multitask environment, respectively. In a normal multitask environment, ready tasks compete for CPU control according to their relative priority. Although you can assign the highest priority (0) to one or more tasks, rescheduling occurs on each system interrupt, or when the executing task issues a system or task call. Thus, in a multitask environment, even the highest priority task may be suspended. Under some circumstances, you may want a task to retain CPU control continuously. To give a task such control, RDOS provides the task call .SINGL.

When a task issues .SINGL, it disables the multitask environment and retains CPU control despite system calls and most task calls that it issues; although interrupts continue, the task Scheduler allows the task to retain control. However, user interrupt routines defined via the .IDEF command continue to execute as usual. The privileged task retains CPU control until it restores the multitask environment by issuing task call .MULTI. The multitask environment is also restored if the task suspends or kills itself.

Generally, a task should not disable the environment unless it must be absolutely autonomous; certainly it should not do so if it relies on other tasks. If you must deny other tasks access to a critical resource, such as a database, use the transmit and receive (.XMT/.REC) mechanism provided by RDOS and discussed earlier in this chapter.

Neither the .SINGL or .MULTI commands affect the other program in a foreground/background environment. As with other task calls, you must declare these two external (.EXTN) in a source program if you want to use them.

.SINGL

Disable the multitask environment

This command disables the multitask environment and gives the issuing task continuing CPU control, despite its priority or any system calls (and most task calls) that it issues. The command is useful for operations outside of user state, as described under "State Definitions" in Appendix I.

Required Input

None.

Format

```
.SINGL
normal return
```

Possible Errors

None.

.MULTI

Restore the multitask environment

This command enables normal scheduler operations and the multitask environment after they have been disabled by the .SINGL command.

Required Input

None.

Format

```
.MULTI
```

Possible Errors

None.

Disabling and Enabling the Task Scheduler

Generally, the RDOS multitask commmands permit you to manage a multitask program with complete satisfaction; the task scheduler always gives CPU control to the ready task of highest priority. In some instances, however, you may want to suspend the task scheduler briefly. You might, for example, suspend rescheduling to control race conditions between several tasks competing for a single resource. This section describes the .DRSCH and .ERSCH commands, which enable you to do so.

Note that disabling the scheduler—even briefly—is a drastic step. The action does not affect system activities such as interrupt service. Moreover, RDOS reactivates the scheduling function as soon as the issuing task loses control of the CPU, even though you may not yet have reenabled rescheduling explicitly. For instance, all system calls, as well as the .SUSP and .KILL commands, reenable scheduling.

.DRSCH

Disable rescheduling

This task call prevents rescheduling in this program environment until you explicitly reenable scheduling or the issuing task loses control of the CPU. Issue task call .DRSCH with caution, since it disrupts the ordinary management of the multitask environment. The task that issues this command retains control even though other, higher priority tasks are ready. This call has no effect when scheduling is disabled.

Required Input

None.

Format

```
.DRSCH
normal return
```

Possible Errors

None.

.ERSCH

Reenable rescheduling

Normally, the task scheduler is enabled and manages the multitask environment within its program. If you have suspended task scheduling by a call to .DRSCH, you can reactivate the scheduler by issuing task call .ERSCH. This call has no effect when scheduling is enabled.

Required Input

None.

Format

```
.ERSCH
normal return
```

Possible Errors

None.

Summary

Table 5.4 summarizes the task, system, and OPCOM commands described in this chapter. Remember that all task names must be declared external via the pseudo-op .EXTN.

Task or System Call	Function
.ABORT	Terminate a task immediately.
.AKILL	Kill all tasks of a given priority.
.ARDY	Ready all tasks of a given priority.
.ASUSP	Suspend all tasks of a given priority.
.DELAY	Delay the caller for the specified number of RTC pulses.
.DQTSK	Dequeue a previously-queued task.
.DUCLK	Define a user clock.
.DRSCH	Disable the rescheduling of the task environment.
.ERSCH	Reenable the rescheduling of the task environment.
.GHRZ	Examine the system real time clock.
.IDST	Get a task's status.
.IOPC	Initialize the Operator Communications Package (OPCOM).
.IXMT	Transmit a message from a user interrupt.
.KILAD	Define a kill-processing address.
.KILL	Kill the calling task.
.MULTI	Enable the multitask environment.
.OVEX	Release an overlay and return to the caller.
.OVKIL	Kill an overlayed task and release the overlay.
.OVREL	Release an overlay node.
.PRI	Change the calling task's priority.
.QTSK	Queue a core-resident or overlay task.
.REC	Receive a message from a task.
.RUCLK	Remove a user clock from the system.
.SINGL	Disable the multitask environment.
.SUSP	Suspend the calling task.
.TASK	Initiate a task.
.TIDK	Kill a task by ID number.
.TIDP	Change the priority of a task by ID number.
.TIDR	Ready a task by ID number.
.TIDS	Suspend a task by ID number.
.TOVLD	Load a user overlay in a multitask environment.
.TRDOP	Read an operator message.
.TWROP	Write an operator message.
.UCEX	Exit from a user clock routine.
.XMT	Transmit a message to another task.
.XMTW	Transmit a message to another task and wait for its receipt.
OPCOM Commands	
DEQ	Dequeue a previously queued task.
KIL	Kill a task.
PRI	Change a task's priority.
QUE	Queue a task for periodic execution.
RDY	Ready a task.
RUN	Execute a task.
SUS	Suspend a task.
TST	Get a task's status.

Table 5.4 System, task, and OPCOM command summary

Chapter 6

Foreground and Background Programming

So far this book has described tools for using RDOS effectively in one program. Chapter 3 explained the essential system calls, Chapter 4 introduced tools for extending memory resources, and Chapter 5 described multitasking; each chapter built upon the features explained in preceding chapters, but all were presented in the context of a single program.

This chapter describes dual programming—the technique of running two, discrete programs simultaneously and letting RDOS apportion CPU and disk I/O time between them.

When you first bootstrap RDOS, only the background is running; the CLI, running in background memory, displays its R prompt. You can then execute a foreground program directly, via the CLI's EXFG command, or you can execute a background program, which in turn may execute another program in the foreground via system call .EXFG.

How you handle dual programming depends largely on whether or not your system has a hardware map to separate the two programs. Dual programming is safer and easier in mapped systems, which offer the added advantages of extended address space described in Chapter 4. If your system is unmapped, you must configure a program for foreground execution by specifying starting ZREL and NREL addresses in the RLDR command line; nonetheless, with a little care, you can execute a program in both an unmapped foreground and background.

This chapter contains the following, major sections:

- Overview
- Dual Programming in Mapped Systems
- Dual Programming in Unmapped Systems
- Foreground/Background System Calls

In a final section, the commands presented in this chapter are summarized in table form. Occasionally a discussion will refer to certain, related commands whose descriptions, in earlier chapters, you may want to refer to. These commands include:

.MEM	Check the current program's NMAX (Chapter 3).
.MEMI	Change the value of NMAX (Chapter 3).
.EXEC	Swap or chain a save file (Chapter 4).
.ERTN and .RTN	Return to the next higher level program (Chapter 4).
.WRPR	Write-protect a memory block, in mapped systems only (Chapter 4).

Overview

The two programs that run under RDOS are called the *foreground* and *background* programs. These programs exist independently of each other, and each has its own task scheduler. The two programs can have equal priority, or you may assign a higher priority to the foreground program. In this case, control goes to the background only when no task in the foreground is ready. When you need to run a real-time program with critical response time, execute it in the foreground. The foreground will then receive the higher priority, while the background can be used for programs not requiring fast response (eg, assemblies, compilations, and the like).

Foreground and background programs can communicate via a Multiprocessor Communications Adapter line, or they can each define a common communications area via the .ICMN command and transmit messages to the other via system calls .WRCMN and .RDCMN. The .FGND command enables the background program to determine whether or not a foreground program exists. The foreground program can terminate itself via .RTN from level 0 (or you can terminate it by typing CTRL-F from the background console), and it can release all its former memory.

Foreground and background programs can access common disk files and common directories. If foreground and background tasks are using the same directory, either task may release that directory without affecting the other task's use of it. If one program, F for example, releases a directory which is in use by program B, F receives the error return with error code EROPD as an indication that the directory is in use by B. Nonetheless, RDOS releases the directory from F.

The foreground and background cannot use the same reserved device file simultaneously; nor can they spool data simultaneously to the same output device. Only the first ground to open the reserved device request will be able to use that device. Similarly, foreground and background programs should not issue simultaneous read commands to a common input device, since RDOS has no way to separate elements in an input data stream and divert them to two different programs.

If you have a mapped system, you can use all mapped system and task calls. RDOS treats any special mapped calls issued in an unmapped environment as no-ops, and gives control to the call's normal return.

Dual Programming in Mapped Systems

Mapped systems provide an absolute hardware boundary between the foreground and background programs. Moreover, the map provides both programs with a complete page zero (including auto increment/decrement locations) and a complete NREL memory area. You can run two CLIs concurrently in a mapped environment, if two consoles are available.

In mapped systems, all programs may use locations 16_8 and above, up to the limits of available memory, since each program has its own page zero. The system initially allots all memory blocks to the background program. You can change the initial memory allocation via the CLI's SMEM command, and can check the current memory allocations via CLI command GMEM or system call .MEM. Each program can change its own NMAX value via system call .MEMI.

Whenever a map violation occurs in an instruction that is not a call (eg, an infinite defer, illegal address, or illegal attempt to reference a system device), RDOS outputs the contents of the program counter and accumulators as follows:

TRAP PC AC0 AC1 AC2 AC3

PC gives either the location of the instruction that caused the trap, or -1 if RDOS is unable to report a meaningful address. You might receive -1, for example, if your program tried a seriously illogical operation such as existing from a user interrupt routine (.UIEX) when no such routine had been defined.

Following its TRAP message, RDOS creates a break save file (named BREAK.SV); places it in the current directory; and displays the message BREAK on the console. Control then goes to the next higher level program in which location USTBR of the UST is set to a valid address. (See Chapter 3 under "Keyboard Interrupts.")

If you pass an illegal address to a system call, RDOS returns error code 74, ERMPR.

Writing interrupt routines for special user devices is slightly easier in a mapped system. If you want a user device to use the data channel, however, you must identify the device via system call .STMAP, described in Chapter 7.

When your program issues a .MEMI command in a mapped environment, RDOS sets NMAX at whatever value is required by the specified memory increment, up to the highest memory address (HMA) available to your program. Nonetheless, the map always allocates memory in blocks of 2000_8 words. Thus, for example, if NMAX is set to 4000_8 and you request a memory increment of 500_8, NMAX becomes 40500 even though a total of 42000 memory words are reserved for the program.

You can build foreground save and overlay files for either ground in a mapped system in the same way that you would for a single-program background, since RDOS reserves an entire ZREL and NREL memory for each ground.

Executing Dual Programs

The RDOS system bootstrap operation brings the CLI into execution in the background. At this point, when the foreground program has yet to be loaded, all available memory is allocated to the background. Thus, before you can issue any foreground command on a mapped machine, you must allocate memory to the foreground with the SMEM command.

After you have built an executable foreground save file (with optional overlays), you can load and execute in the foreground area by entering the CLI's EXFG command followed by the save file's name and a terminator. (Any background program can also execute a program in the foreground by issuing system call.EXFG.)

You can issue the EXFG or .EXFG commands for any executable program, including a system utility or the CLI itself, and access it via a second system console, $TTI1/$TTO1. (If you use system call .EXFG instead of its CLI counterpart—a utility command—you must set up the fore-

ground command file, FCOM.CM, as described in the manual *RDOS/DOS Command Line Interpreter.*)

To execute a single-system utility program in the foreground, issue the following command from the background console:

```
EXFG system-utility-command-stream ⟨CR⟩
```

For example, to assemble source file ABC in the foreground with a cross reference and listing to the line printer, you would type:

```
EXFG MAC ABC $LPT/L ⟨CR⟩
```

To execute the CLI itself or any other save file in the foreground, use the form:

```
EXFG programname ⟨CR⟩
```

Any program executing in the foreground may push other program levels into execution via system call .EXEC.

The foreground program can terminate by issuing as many .RTN (or .ERTN) commands as needed to pop through level 0 (if the CLI is not active in the foreground). This occurs when a single-system program, executed at level 0 in the foreground, terminates its operation. Alternatively, you can terminate a foreground program by typing CTRL-F on the background console. You must use this second method to terminate a program that incorporates a CTRL-A or CTRL-C handler. When you issue CTRL-A or CTRL-C via the foreground console (if any), the foreground program terminates if (1) RDOS finds no interrupt processing address in USTIT/USTBR of the foreground UST, and (2) no higher level program contains such a processing address in its UST. Each system utility automatically issues a .RTN command when it terminates to return control to the background (or, if executing in the background, to return control to the CLI).

Whenever the foreground program terminates via system calls .RTN or .ERTN, RDOS displays the message

```
FG TERM
```

on the console. The same message appears if you terminate the foreground with a CTRL-F interrupt.

Checkpointing a Background Program

Checkpointing allows a foreground program to interrupt the current background program, run a new program in the background, and then restore the original background program.

Some processing applications function more effectively if the foreground program can make use of the background's resources in this way. One example of such an application is a mapped, dual-program system containing a data collection program in the foreground and one of several system utilities in the background. In such an application, the foreground might occasionally need to execute a data reduction program in the background. Checkpointing the data reduction program into execution from time to time would fulfill this need. You can checkpoint via the mapped RDOS system call .EXBG, described in this chapter.

Dual Programming in Unmapped Systems

Unmapped systems must use software boundaries to separate the foreground and background program areas. You must define these boundaries before execution, in the RLDR command line.

Each boundary is a starting address for execution; the local /F switch defines the starting NREL address, and the /Z switch defines the starting ZREL address for execution. Locations 20_8 through 37_8 are reserved for use by the background.

Building Foreground Programs

When you plan to run foreground and background programs in an unmapped system, bear in mind that the memory requirements of each will be critical. Aside from this factor and any foreground/background system calls that you plan to use, writing the source code for a foreground program does not require special consideration.

Depending on your application, you may want a *background* program to change NMAX (.MEMI, Chapter 3) if it will execute a specific program in the foreground via system call .EXFG.

After writing and assembling your source program, configure it for foreground operation by including the starting ZREL and NREL addresses in the RLDR command line. (Adopt the practice of checking the ZMAX and NMAX requirements of programs that you may want to execute simultaneously in the background; you can do this with the program load map or with the SEDIT utility.) The software boundaries must include both NREL and ZREL address information in local switches F and Z, for example:

```
RLDR 13000/F 250/Z R0 R1 [ OV0 OV1, OV2 ]
```

This command line creates a save file named R0.SV (containing binary files R0 and R1), and an overlay file named R0.OL (containing two overlays). When you load the save file into memory, RDOS loads its ZREL portion into locations 250_8 and above, and its NREL portion into locations 13016_8 and above.

When building programs for an unmapped foreground, remember that they will be separated by soft boundaries only; hardware does not protect the address space of the two programs. Thus, for example, you must ensure that no background program attempts to return to a higher-level, background program requiring more core storage. If such a return is performed (eg, via .RTN) and the larger background program requires space now occupied by the foreground program, system failure results. This situtation would occur after the following sequence of program loads:

1. The CLI resides in the background, and no foreground program is executing.

2. A background program named BGD, smaller than the CLI, is executed via the CLI on level 1.

3. BGD issues the foreground load command, .EXFG, loading a larger program whose starting address immediately follows BGD's NMAX.

4. BGD issues the .RTN command, attempting to return to the CLI. The CLI, however, requires memory storage which the foreground program now occupies. System failure occurs.

You can avoid this mistake by planning your program flow with care.

Executing Dual Programs

The RDOS bootstrap operation brings the CLI into execution in the background. At this point, when the foreground program has yet to be loaded, all memory is allocated to the background. Once you have built an executable save file, you can execute it in the foreground via the CLI's EXFG command or with system call .EXFG. For either command to work, you must have loaded the foreground program with information about its software boundaries.

To load and execute a program in the foreground, type the command line:

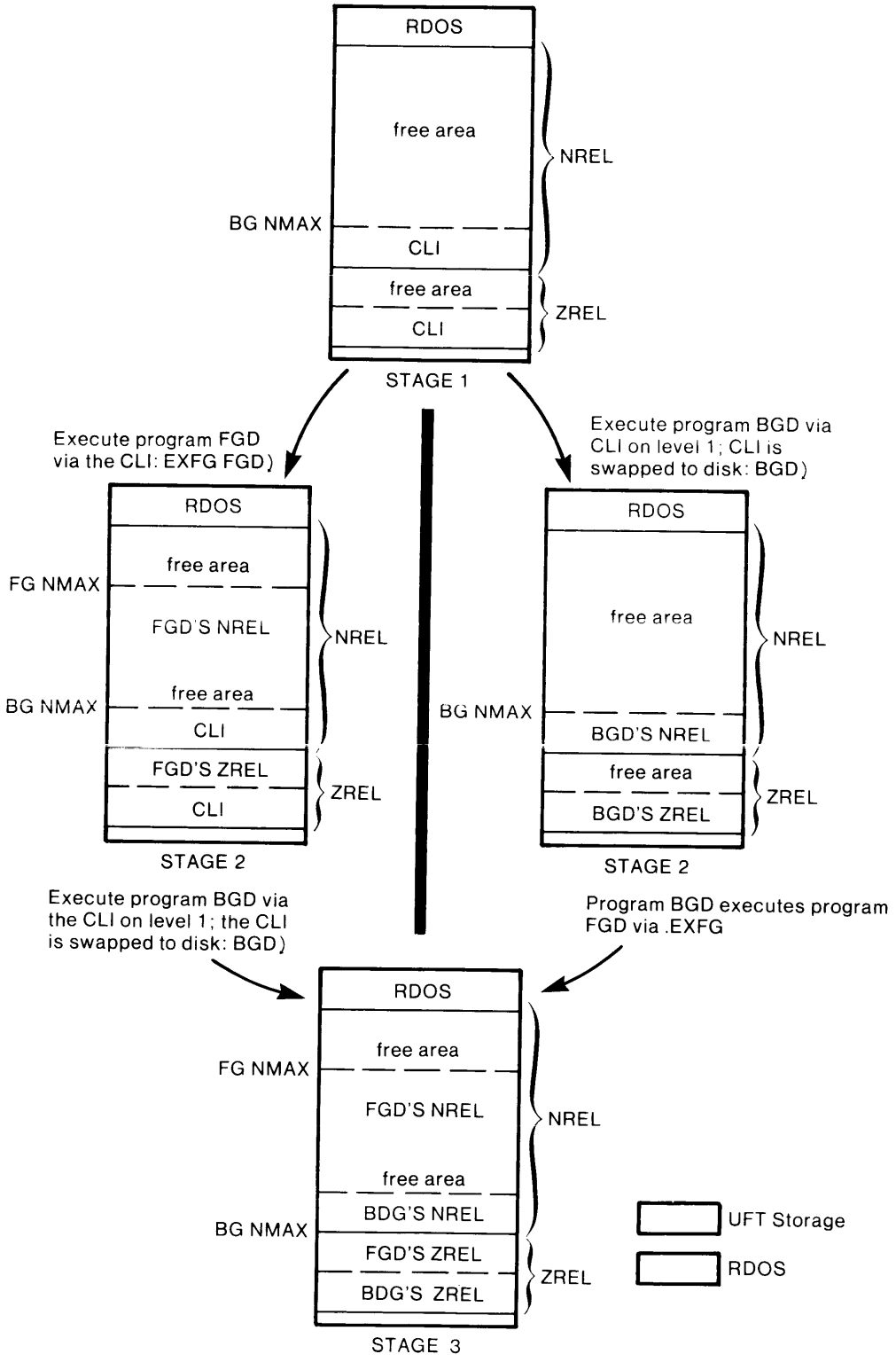
```
EXFG programname (CR)
```

If the boundary requirements of this program threaten to overwrite any portion of the CLI or background program, RDOS will not load the foreground program. Otherwise, if its boundaries are valid, RDOS loads and executes the foreground program; the CLI displays its R prompt when execution begins. You can then try to execute a new background program via the CLI, thereby swapping the CLI to disk. If the program you wish to execute in the background requires more memory than is available, RDOS does not execute it.

You can terminate a foreground program by typing CTRL-F on the background console, or CTRL-A (or CTRL-C) on the foreground console, $TTI1 (if any). Any of thos actions terminate a foreground program as long as (1) it has no interrupt processing address in USTIT (or USTBR) in its UST, and (2) no higher-level foreground program has such a processing address in its UST. The foreground program can release its memory to the background by issuing a .RTN command.

When you terminate the foreground via CTRL-F, CTRL-A, or CTRL-C, or when the foreground program yields its memory to the background via a .ERTN or .RTN command, the message FG TERM appears on the background console.

Figure 6.1 depicts two possible command sequences to produce foreground/background operation in an unmapped system. It uses two sample programs, FGN and BGD. Shaded areas represent storage areas occupied by User File Tables (UFT's). These are 45_8-word structures used by the operating system to record file and device information for each disk file opened on a channel. RDOS stores file information in a section of each UFT called a UFD; you can access UFD information with the .STAT command, described in Chapter 3. In all mapped systems, UFTs reside in system space.



To return to stage 1, FGD issues RTN, relinquishing its memory to BGD. BGD then issues .RTN. From the console, typing CTRL-A, then CTRL-F would achieve the same end by interrupting the programs.

Figure 6.1 Loading foreground and background programs in an unmapped system

SD-00531

Foreground/Background System Calls

This section describes the system calls used to implement dual programming. In order of discussion, they are:

.EXFG Execute a program in the foreground.

.FGND See if the foreground is running, and check the status of the current program.

.ICMN Define a program communications area.

.WRCMN Write a message to the other program.

.RDCMN Read a message from the other program

.WROPR Write an operator message.

.RDOPR Read an operator message.

.EXBG Checkpoint a mapped background program

.EXFG

Execute a program in the foreground

This call loads a program save file into foreground memory and transfers control to it. Only a background program can issue this command. In an unmapped system, you must have loaded the save file with boundary information as explained in the preceding section. RDOS passes the contents of AC2 to the foreground program.

Required Input

AC0 - Byte pointer to the foreground program save file's name.

AC1 - Appropriate starting address/foreground priority code. Two possible addresses are allowed: the program starting address (USTSA), and the Debug III starting address (USTDA). The codes permitted in AC1, and their meanings, are:

0B15	USTSA. Pass control to the ready task of highest priority in the program. (Initially this is the program itself.)
1B15	USTDA. Pass control to the debugger.
0B1	Give the foreground program a higher priority than the background.
1B1	Give the foreground and background the same priority.

Format

```
.SYSTM
.EXFG
error retrn
normal return
```

Possible Errors

AC2	Mnemonic	Meaning
1	ERFNM	Illegal filename.
4	ERSV1	File requires Save attribute.
12	ERDLE	File does not exist.
21	ERUFT	Mapped systems only: not enough channels defined during system generation to satisfy the value specified in USTCH of the save file.
26	ERMEM	Attempt to allocate more memory than is available.
32	ERADR*	Illegal starting address.
53	ERDSN	Directory specifier unknown.
66	ERDNI	Directory not initialized.
70	ERFGE	Foreground already exists.
73	ERUSZ	Too few channels defined at load time or during system generation.
74	ERMPR	Address outside address space.
101	ERDTO	Disk timeout occurred.

*RDOS returns ERADR if the code input in AC1 is illegal or if the required address is missing from the UST. This can occur if (1) you did not specify a starting address for the save file and you input code 0B15 in AC1, or (2) you did not load the debugger as part of the save file and you input code 1B15 in AC1.

.FGND

See if a foreground program is running and check your own level

This system call is used to determine whether or not a foreground program is running in the system, and at what level the calling program is running. The command passes -1 in AC0 if it finds a foreground program, and passes 0 in AC0 if it does not. In AC1, the .FGND command returns a code indicating the calling program's level. The possible codes and their meanings are:

1 Background level 0

2 Background level 1

3 Background level 2

4 Background level 3

5 Background level 4

6 Foreground level 0

7 Foreground level 1

10 Foreground level 2

11 Foreground level 3

12 Foreground level 4

Required Input

None.

Format

```
.SYSTM
.FGND
error return
normal return
```

Possible Errors

None.

.ICMN

Define a program communications area

This system call permits your program to define a contiguous area of up to 256_{10} words within its own address space to send or receive messages from another program. The foreground and background may each define one communications area.

Required Input

AC0 - Starting address of the communications area.

AC1 - Size of the communications area in words.

Format

```
.SYSTM
.ICMN
error return
normal return
```

Possible Errors

AC2	Mnemonic	Meaning
62	ERCMS	Communications area exceeds the program size or would overwrite the system.
74	ERMPR	Address outside address space.

.WRCMN

Write a message to the other program

This system call writes a message of up to 256_{10} words from the calling program (foreground or background) into the other program's communication area. The message sent may originate from anywhere within the sender program's address space.

Required Input

AC0 - Word address of the start of the message.

AC1 - Word offset within the other program's communications area which will receive the message.

AC2 - Number of words to be sent.

Format

```
.SYSTM
.WRCMN
error return
normal return
```

Possible Errors

AC2	Mnemonic	Meaning
62	ERCMS	Message too large for communications area.
63	ERCUS	No communications area is defined in the other program.
74	ERMPR	Address outside address space.

.RDCMN

Read a message from the other program

This system call allows the calling program to read a message of up to 256 decimal words from another program's communications area. The receiving program may accept the message anywhere within its address space.

Required Input

AC0 - Starting word address to receive the message.

AC1 - Word offset within the other program's communications area where the message originated.

AC2 - Number of words to be read.

Format

```
.SYSTM
.RDCMN
error return
normal return
```

Possible Errors

AC2	Mnemonic	Meaning
62	ERCMS	The size of the requested message exceeds the size of the communications area.
63	ERCUS	No communications area is defined in the other program.
74	ERMPR	Address outside address space.

.WROPR

Write an operator message

This system call instructs the calling program to write a text string to the system console, $TTO. There may be only one outstanding write-operator command in a program area. The message must consist of an ASCII string less than or equal to 129 characters in length, including a carriage return, form feed, or null terminator. On the console, RDOS displays two exclamation points (!!), either an F or B, and then the message. Letters F or B indicate whether the message came from the foreground or background program, respectively. Thus, text strings appear on the console are in one of two forms:

!!Ftext string or *!!Btext string*

You should not issue this call if you have also used OPCOM commands or task calls .TWROP and .TRDOP in the environment.

Required Input

AC0 - Byte pointer to text string.

Format

```
.SYSTM
.WROPR
error return
normal return
```

Possible Errors

AC2	Mnemonic	Meaning
74	ERMPR	Address outside address space.
120	EROPM	Operator messages not specified during system generation.

.RDOPR

Read an operator message

This system call prepares the calling task to receive an operator message from the system console, $TTI; the task may exist in either the foreground or background programs.

Before typing the message to the program, you must type CTRL-E (echoed on the console as !), followed by an F or a B to indicate whether a foreground or background program is to receive the message. RDOS recognizes CTRL-E only if it is the first character in a line.

If no program has requested a console message, the TTY bell (if any) rings when you press CTRL-E; if the second character is any other than an F or B (or rubout), the TTY bell rings again and RDOS accepts no further input until you type the correct character.

If immediately after pressing CTRL-E you wish to cancel the message transmission, press the RUBOUT key instead of characters F or B. This key erases message characters, starting with the most recent one typed. RDOS echoes a left arrow (←) on teletypewriters; on CRT displays, it erases the last character each time you press the key. The last character in the message string must be a carriage return, form feed or null, and the total length of the message, including its terminator, can be up to 132 characters.

Only one program (task) in each ground may have a read-operator message request outstanding at any given moment. The .RDOPR command should not be issued if you are using OPCOM commands or task calls .TWROP and .TRDOP in this program environment.

Required Input

AC0 - Byte pointer to message area.

Format

```
.SYSTM
.RDOPR
error return
normal return
```

On the normal return, RDOS passes the message byte count (including the terminator) in AC1.

Possible Errors

AC2	Mnemonic	Meaning
74	ERMPR	Address outside address space.
120	EROPM	Operator messages not specified during system generation.

.EXBG

Checkpoint a mapped background program

Checkpointing is the practice of suspending one background program (the checkpointed program) temporarily so that you can execute a new program in the background. Only a mapped, foreground program may issue the checkpoint call. The foreground can also pass an optional, one-word message to the new background program. There may be only one checkpointed program at a time; RDOS does not allow nested checkpoints.

Before you can checkpoint a background program, it must meet two conditions: (1) it must not perform any multiplexor I/O, and (2) it must not use any of the following system calls:

.DELAY
.RDOP
.IDEF/.IRMV
.DUCLK/.RUCLK

When a background program is checkpointed, RDOS displays the message

CP ENT

on the console. RDOS saves the following constants from the original background program, and restores them when it restores that program:

priority
floating-point processor state
ongoing console input
current directory.

During the checkpoint, the current directory for both grounds is the foreground's current directory. Thus, if the new background program needs to access files, they must be in the current directory or you must include directory specifiers to them.

The new program can be assigned one of two priorities: that of the foreground, or that of the checkpointed program.

Since RDOS preserves $TTI input to the checkpointed program, $TTI becomes unavailable for use by the new background program except via the .RDOP command. The new program can direct output to $TTO via system call .WROP, which writes an operator message.

The new program can restore the checkpointed program by issuing the .ERTN or .RTN commands; you can also restore the checkpointed program by entering CTRL-A or CTRL-C from $TTI, provided the new program's UST does not specify a different interrupt routine.

On a keyboard interrupt, RDOS displays the message

CP INT

on $TTO. It displays the message

CP RTN

on $TTO when the new program restores the checkpointed program normally, via a .RTN or .ERTN command.

Required Input

AC0 - Byte pointer to the new background save file's name.

AC1 - 1B0: give the new background program the same priority as the checkpointed program. Clear all other bits in AC1 to zero.

AC2 - Optional, one-word message to the new background program.

Format

```
.SYSTM
.EXBG
error return
normal return
```

Possible Errors

AC2	Mnemonic	Meaning
1	ERFNM	Illegal filename.
2	ERICM	Attempt to checkpoint in an unmapped system.
4	ERSV1	File requires S (save) attribute.
12	ERDLE	File does not exist.
21	ERUFT	Not enough channels defined during system generation to satisfy the value specified in USTCH of the new background program.
25	ERCM3	Attempt to checkpoint a checkpointed background program.
26	ERMEM	Attempt to allocate more memory than is available.
53	ERDSN	Directory specifier unknown.
57	ERLDE	Link depth exceeded.
66	ERDNI	Directory not initialized.
73	ERUSZ	Too few channels defined at load time or during system generation.
74	ERMPR	Address outside address space.
76	ERNTE	Program to be checkpointed is not checkpointable, or attempt to create two outstanding checkpoints.
101	ERDTO	Disk time-out occurred.

Example

In the following sequence, three sample programs complete a checkpoint procedure. The background program to be suspended with a checkpoint is called BACK, the foreground program that will execute the checkpoint is called FORE; and the program to be checkpointed into the background is named COMP.

1. First the programs FORE and BACK are executed from the CLI:

```
EXFG FORE ⟨CR⟩
R

EXFG BACK ⟨CR⟩
```

2. While both FORE and BACK are running, FORE issues the .EXBG command to COMP, checkpointing COMP into execution. BACK is suspended, but RDOS saves its current state, the FPU, all $TTI input to it, and remembers its current directory. The console displays the message *CP ENT.*

3. COMP reads data from some of FORE's files; it issues a few .WROP and .RDOP commands and receives replies from the console.

Having done its work, COMP writes data to a file in FORE's current directory. It then signals FORE that it is done via call .WRCMN. FORE receives the message, reads COMP's data from the file, and continues.

4. COMP issues system call .ERTN. The console displays *CP RTN.* And BACK resumes execution from its original, current directory. The console displays *CP RTN.*

Summary

Table 6.1 summarizes the system calls for dual programming discussed in this chapter.

System Call	Function
.EXBG	Suspend one mapped, background program and execute another.
.EXFG	Load a program save file into foreground memory and execute it.
.FGND	See if a foreground program is running, and check the program level at which the caller is running.
.ICMN	Define a communications area.
.RDCMN	Read a message from another program's communications area.
.RDOPR	Read an operator message from the system console, $TTI.
.WRCMN	Write a message from the calling program (foreground or background) to the other program's communications area.
.WROPR	Write a text string to the system console, $TTO.

Table 6.1 System call summary

Chapter 7

Interrupts and Power Failures

Many real-time computer control systems require interaction with nonstandard devices to obtain information from, and provide information to, real world environments. This chapter explains how to service interrupts from devices, how the system handles power failures, and how you can write user power-fail routines.

Servicing User Interrupts

This section includes

- Commands for Interrupt and Power Fail Routines
- Generalized I/O Routines
- I/O Buffer Module

When the CPU detects an interrupt request, it suspends the current program and directs control to its device interrupt service program, INTD. (INTD is part of RDOS and is always memory-resident.) The CPU then directs control through the interrupt vector table to the proper device control table (DCT), using the device code as a guide.

After a user interrupt occurs, control goes to your service routine; AC3 contains the return address required for exit from your routine, and AC2 contains the address of the DCT. The task call .UIEX exits from the routine and returns to the current environment. You can issue .UIEX in both single- and multitask environments.

RDOS removes all user devices from the system when either a program swap or a chain occurs. When the system receives a user interrupt on a program level that has not identified the user device, it issues an NIOC to the device and then returns to normal program execution.

Whenever a device requiring special user service generates an interrupt request, the entire task environment halts until RDOS has serviced the interrupt. All tasks resume their former states when the environment restarts, unless you transmit a message to one of them via the .IXMT command from the interrupt service routine. (The .IXMT command was discussed in Chapter 5.) Rescheduling of the program and task environment can occur upon return from the routine, depending on the contents of AC1 in the return command. (See .UIEX, described next.)

Commands for Interrupt and Power Fail Routines

In addition to the .IXMT command, your user interrupt or user power fail routine can issue task calls .SMSK, .UIEX, and .UPEX. These commands, along with system calls .IDEF, .IRMV, and .STMAP, are described in this section. They apply to both single- and multitask environments, and, unless otherwise noted, to both mapped and unmapped machines.

.IDEF

Identify a user interrupt device

This system call introduces to RDOS a device that you did not identify during system generation, but whose interrupts you want the system to recognize. (The .IDEF call places an entry in the interrupt vector table.) A maximum of 10 user devices can be identified to the system at any moment. An .IDEF to any device also provides access to device code 77_8, so that you can do such things as disable and enable interrupts.

The number of free device codes (those that you can assign to user devices) depends on the hardware in your RDOS system. You can find system devices and their codes on the instruction reference card for your computer.

If your system has an IPB, and you want control when the watchdog timer times out, you must identify the timer via the .IDEF command. (See Chapter 8.) If you generated the current RDOS system without an IPB and subsequently introduce a device on device code 36, RDOS issues a NIOP to device code 37 whenever the real-time clock or power-fail monitor interrupts. (The IPB has device code 36, and the watchdog timer, device code 37.) To prevent this interaction from occurring, avoid using device codes 36 or 37 for a user device.

To introduce a data channel device, your program must establish the data channel map for the device via the .STMAP command discussed later in this section. (The .STMAP command applies to mapped systems only.)

If you introduce communication software such as CAM, RDOS throws away interrupts left outstanding from programs that terminate without clearing their devices.

Required Input

AC0 - Device code of the new device.

AC1 - Address of the new device's DCT. In a mapped system, this address must be in NREL space, ie, above 400_8. Also in a mapped system, set bit 0 to 1 if you want the new device to use the data channel.

AC2 - Mapped systems only: number of 1K core blocks required by the data channel map. This number must be one larger than the integer number of 1,024-word blocks used for data channel core buffers. (Applicable only if you have set bit 0 of AC1 to 1 for this call.)

Format

```
.SYSTM
.IDEF
normal return
error return
```

Possible Errors

AC2	Mnemonic	Meaning
36	ERDNM	Illegal device code (greater than 76_8). Device code 77_8 is reserved for CPU which supervises the power monitor/ auto restart option.
45	ERIBS	Interrupt device code in use, or 10 user devices already identified.
65	ERDCH	Unmapped systems only: insufficient room in data channel map.
74	ERMPR	Mapped systems only: address outside address space.

.UIEX

Exit from a user interrupt routine

This command returns control to a program environment after a user interrupt; you can use it only to terminate an interrupt service routine. In all systems, you can force rescheduling by passing a nonzero value in AC1; if AC1 contains 0 when you issue this command, the environment resumes without rescheduling. In a mapped system, RDOS ignores values input in the other accumulators. In an unmapped system, you must restore AC2 and AC3 to the addresses they held on entry to the routine; otherwise, the system will crash.

Required Input

AC1 - Zero only to suppress rescheduling.

AC2 - Unmapped systems only: address upon entry to routine (DCT).

AC3 - Unmapped systems only: address upon entry to routine (return address).

Format

```
.UIEX
```

Possible Errors

None.

.UPEX

Exit from a powwer fail service routine

This command accomplishes an exit from a user power fail service routine, forcing rescheduling. Control returns to the location that was interrupted by a power failure. The same restrictions applying to system and task calls in a user interrupt service routine apply to a user power fail routine. The .UPEX command is discussed again in the context of power fail/auto restart procedures at the end of the chapter.

Required Input

AC3 - Return address upon entry to the routine (unmapped systems only).

Format

```
.UPEX
```

Possible Errors

None.

.IRMV

Remove a nonSYSGENed interrupt device

To prevent the system from recognizing an interrupt device that was identified by the .IDEF command, issue system call .IRMV.

Required Input

AC0 - Device code for the device that you want to remove from the system.

Format

```
.SYSTM
.IRMV
error return
normal return
```

Possible Errors

Only one possible error results from this command. Its mnemonicis ERDNM, indicating an illegal device code (greater than 77_8) or an attempt to remove a SYSGENed device. RDOS returns code 36 in AC2 when this error occurs.

.SMSK

Modify the current interrupt mask

Use this task call to change your interrupt mask for a service routine in both single- and multitask environments. Whenever a user interrupt occurs, RDOS ORs the interrupt mask with the mask in the DCTMS of your DCT to produce the current interrrupt mask. The .SMSK command allows your interrupt routine to change the old mask and produce a new one which is the logical OR of the old mask and a new value. The .SMSK command destroys the accumulators, so you must restore them for the subsequent exit via task call .UIEX.

Required Input

AC1 - New value to be ORed with old mask.

Format

```
.SMSK
normal return
```

Possible Errors

None.

.STMAP

Set the data channel map

Before a user device can employ the data channel in a mapped system, your program must issue the .STMAP command to set up the data channel map. This is a special map maintained by the mapping hardware for data channel use. The .STMAP command sets up the data channel map for the user device and returns in AC1 the logical address that you should send to the device. This call is a no-op when issued in an unmapped system.

Required Input

AC0 - Device code.

AC1 - Starting address (in your address space) of the device buffer.

Format

```
.SYSTM
.STMAP
error return
normal return
```

Possible Errors

AC2	Mnemonic	Meaning
36	ERDNM	Device code not previously identified via .IDEF as a data channel device.
74	ERMPR	Address outside address space.

Power Fail/Auto Restart Procedures

RDOS provides software support for the power fail/automatic restart option. When the system detects a power loss, it transfers control to a power fail routine that saves the status of all accumulators, the PC, and Carry.

If the console key is in the LOCK position when power returns, the system console displays this message once power is restored:

POWER RESTORED

If possible, the system restores task state variables, resuming operating at the point of interruption. After this message appears, your disk drives may require extra time (up to one minute) to come back on line.

If the console key is in the ON position when power returns, you must set all data switches to zero (down) and lift START once power is restored. The message POWER RESTORED is then displayed; task state variables are restored, and operation resumes.

RDOS provides power-up restart service to the following system devices:

Teletypewriters and CRTs
Disks
Multiplexors
Line printers
Paper tape readers and punches
Card readers
Plotters

Character output devices may lose one or more characters during power up. Since power-up service for disks includes a complete reread or rewrite of the current disk block, you will lose no disk information, although you must wait for the disk unit's READY indicator to light. When power returns, RDOS restores modem multiplexor lines when the user dials in. Line printers may lose up to a single line of information. Card readers may lose up to 80 columns of information on a single card. Devices requiring operator intervention, such as line printers, must receive an operator's attention if power was lost for an extended period of time.

Note that RDOS does not provide power-up service for magnetic tape units, and that no power-up service is possible for semiconductor memory without a backup battery.

Power-up Service for User Devices

To provide power-up service for a magnetic tape unit or for your own device, you must write an interrupt service routine using the .IDEF command as follows.

Required Input

AC1 - Starting address of the user power-up service routine.

Format

```
.SYSTM
.IDEF
error return
normal return
```

The error return is never taken.

In both mapped and unmapped systems, exiting from a user power-up service routine forces rescheduling and is accomplished by task call .UPEX. The same restrictions applying to the use of system and task calls in a user interrupt service routine apply to a user power fail routine.

Upon entering a user power fail service routine, AC3 contains the address required to exit from it. To return from the routine in an unmapped environment, AC3 must be loaded with this return address, and task call .UPEX must be issued. In mapped systems, the value input in AC3 when this call is issued is ignored. Issue the .UPEX command according to the following guidelines.

Required Input

AC3 - Return address upon entry to the routine (unmapped systems only).

Format

```
.UPEX
```

Control returns to the location which was interrupted by a power failure. No error or normal returns need be reserved. The .UPEX command can be issued in a single-task environment. Note that this command applies only to revisions 03 and higher of RDOS.

Summary

This chapter described several system and task calls that figure importantly in interrupt and power fail programs. These commands are summarized in Table 7.1.

Command	Function
.IDEF	Introduce to RDOS a device, not defined during system generation, whose interrupts you want the system to recognize.
.IRMV	Prevent the system from recognizing a device defined via the .IDEF command.
.SMSK	Modify the current interrupt mask.
.STMAP	Set up the data channel map for a user device.
.UPEX	Exit from a power fail service routine.
.UIEX	Exit from a user interrupt routine.

Table 7.1 System and task call summary

Chapter 8

Multiple Processor Systems

This chapter describes managing a system that includes more than one Data General computer. There are two hardware options available to manage such a system: (1) an Interprocessor Buffer (IPB), Model 4240, which allows two CPUs to communicate via full duplex lines; and (2) a Multiprocessor Communications Adapter (MCA), Model 4206, which allows up to 15 CPUs to communicate via full duplex lines. The MCA also allows foreground and background programs to communicate at data channel speeds.

Overview

If you have an IPB or MCA, you can run your processors together, in a multiprocessor system; this system can use any or all of the features described in previous chapters of this book. If you have neither device, each CPU in your installation must run independently.

You can configure an RDOS system to support either (or both) an IPB or MCA during system generation by correctly answering the questions that SYSGEN asks these devices.

The IPB provides one, full-duplex line for sequential and line I/O between two processors. It also provides a half-duplex line for RDOS; this enables RDOS to assure that systems sharing disk partitions do not simultaneously modify the system (SYS.DR) or map (MAP.DR) directories of any partition. The IPB also provides an interval timer that permits each processor to monitor the activity of the other. If either processor fails to service its real time clock periodically, the timer alerts the other processor.

Note that on a hardware level, each shared disk must be installed with the same device code and unit name. If one processor has a disk hard-wired as the first controller, DP0, the second processor must also have the disk hardwired as the first controller, DP0. Both processors reference the disk as DP0. If a disk is unshared, however, its device name must be unique.

Both processors must run with an RDOS of the same revision level for IPB support to work. If not, one processor or the other will very likely enter Exceptional Status, a condition described in Appendix E.

IPB support maintains the integrity of system files and disk file structures, but does not provide protection for the contents of user files. Thus, if both users try to access the same file simultaneously one file, or fractions of it, may be lost.

A typical IPB system consists of two CPU's operating independently. This system permits each CPU to have a foreground and background program; programs in both CPUs can access files in the same disk partition. The IPB maintains the integrity of SYS.DR and MAP.DR in common disk partitions, and allows each CPU to monitor the other's activity.

Another dual-processor application might use the IPB to back up a critical, real-time program. In critical real-time situations, redundancy helps safeguard the total system, and allows it to continue running even if a CPU fails. One example of a fail-safe IPB application is a main system that runs the critical process, while a back up system stands ready to assume the main system's functions should it fail. While it is standing by, the back-up system runs jobs of lower priority, such as data analysis, summary reporting, and program development. If the main system fails, the interval timer detects this failure and signals the back-up system to take control.

The MCA does not have an interval timer; nor does it allow CPUs to share disk directories. It does, however, enable up to 15 CPUs to communicate via their data channels. Each MCA controller supports up to 15 separate lines, and each MCA line provides asynchronous, full-duplex communications links for sequential I/O. Each line is a filename, which your program can access via system calls and which you can access via CLI commands. You can also transmit an entire RDOS system via the special CLI command, MCABOOT. Each MCA line offers high-speed, interprogram or interprocessor communications with little processor overhead.

RDOS itself does not use the MCA. Unless you generate RDOS with IPB support, it does not maintain the integrity of a partition accessed by more than one processor.

To run a multiprocessor system under either IPB or MCA, each CPU must bootstrap an operating system in a seperate disk partition, and each partition must have its own copy

of an RDOS system and CLI (files CLI.SV, CLI.OL, and CLI.ER).

Interprocessor Buffer (IPB) Programming

This section discusses IPB programming considerations such as the interval timer and full-duplex, communications line featured IPB hardware. The section also examines a hypothetical IPB program, and describes system call .BOOT, which bootstraps the separate operating systems required in multiprocessor environments.

Interval Timer

The Interprocessor Buffer (IPB) hardware features an interval timer that tells one processor when the other processor has stopped. Specifically, the timer generates an interrupt request if either processor fails to service its real time clock every second. RDOS treats this interrupt request as a user interrupt. You can write routines to identify the interrupt via system call .IDEF, described in Chapter 7. The device code of the interval timer is 37_8.

An interval timer interrupt indicates to RDOS that the other processor has stopped; hence, you should not use IDEB, or any other program that suspends interrupts for extended periods, while both processors are running.

Dual Processor Program Communications

IPB hardware also allows two processors to communicate via a full-duplex line. This communications link permits a user program running in either processor to read or write line or sequential I/O to the other processor, via special filenames. The filenames for the read and write operations are:

$DPI - Input dual processor link (device code 40_8).

$DP0 - Output dual processor link (device code 41_8.)

Each side has links $DPI and a $DPO. The output link ($DPO) of each side is connected to the other side's input link ($DPI). Thus, one side's $DPO writes to the other side's $DPI. To show how this scheme works in practice, assume two CPUs named CPUA and CPUB, and their respective programs, PROGA and PROGB. If PROGA wants to write to PROGB, it would do so via output link $DPO; PROGB would read from input link $DPI. Simultaneously, PROGB could write a message to PROGA via its own output link. Each $DPO is a spoolable device. (PROGB should issue the read request before PROGA issues the write, or a character will be lost.)

IPB Example

In this example, the main program (P) monitors and controls a real-time environment, and a secondary program (S) stands by to take over if P fails. A special restart task will bootstrap a system for S via system call .BOOT, described next.

P, the control program, runs in the foreground of one CPU, while less critical programs run in the background. (P could also run in the background of a single-ground environment.)

As the primary program (P) monitors and controls the real-time environment, it sends periodic status reports to a log file on its disk so that, in the event of its failure, S can seize control and maintain continuity.

The backup program, S, runs in the second CPU, in either a single- or dual-ground environment. At its very beginning, S creates a highest-priority restart task, which suspends itself by issuing a .REC call to the user's interval timer interrupt routine. This interrupt routine issues the .IXMT command to activate the restart task, which then bootstraps S.

If the main CPU, running P, develops a problem, the interval timer generates an interrupt, and the interval timer service program in S readies the restart task. The restart task then closes all files in program S; releases all of S's directories; resets I/O; and bootstraps a new RDOS system, identical to P's. Having bootstrapped this new system, S reads P's status reports to determine where it stopped, and proceeds to monitor and control the real-time environment.

.BOOT

Bootstrap a new operating system

This command executes an orderly shutdown of the current RDOS system, and bootstraps the system you have indicated by a byte pointer in AC0. Specifically, a .BOOT operation resembles that of the CLI's BOOT command—it closes all background and foreground files; releases their directories; resets all I/O; and then bootstraps the new system, which must exist in a secondary or primary partition. If the byte pointer specifies a link entry to the new system, you must also link the new system's overlay file and initialize all partitions involved in the resolution chain.

When you bootstrap a system conventionally, it asks questions about the date and time, and then invokes the CLI. If all data switches are in the up position or the switch register contains -1, RDOS searches for a file named RESTART.SV and, if unable to find it, invokes the CLI. If a program issues the .BOOT command and all data switches are in the up position or the register contains -1, the new system comes up automatically with the default time and date of January 1, 1968; then .BOOT chains control on level 0 to the file RESTART.SV. If this file does not exist, the .BOOT command asks the conventional log-on questions.

The file RESTART.SV does not initially exist, but must be created in order to bootstrap a system without operator intervention. It could also be the name of the user program itself, or linked to the program name. If the current date and time are important to the real-time process, you must find some way to get them to the new program—perhaps via RESTART.SV itself, if the old program periodically stored date/time data in a file that RESTART can read when it assumes control.

Required Input

AC0 - Byte pointer to name of new operating system.

Format

```
.SYSTM
.BOOT
error return
```

There is no normal return, since upon the normal completion of this call BOOT receives and then passes control to the new operating system.

Possible Errors

AC2	Mnemonic	Meaning
23	ERRTN	File RESTART.SV does not exist, yet data switches were set for restarting without operator intervention.
53	ERDSN	Unknown directory specifier.
74	ERMPR	Address outside address space (mapped systems only).
101	ERDTO	Disk timeout occurred.
107	ERSFA	Spool file is active.

MCA Programming

This section discusses the MCA programming considerations of data transmission, the use of CLI commands on MCA lines, and the transmission of operating systems or stand-alone programs from one MCA unit to another. The section also describes the .GMCA command which retrieves the MCA number of the current CPU.

Data Transmissions

The type 4206 Multiprocessor Communications Adapter receiver/transmitter (MCAR/MCAT) allows programs to communicate over full-duplex lines, in blocks of up to 8192 bytes, via the data channel. Each program can exist within the program space of a single CPU, or within up to 14 other CPUs, or both. A second 4206 receiver/transmitter, MCAR1/MCAT1, provides up to 15 more communications links. Each CPU may communicate with any other CPU.

Depending on whether it is transmitting or receiving, each MCA line is a filename of the following form:

MCAT(1):rr
or
MCAR(2):tt

where rr represents a receiver unit number from 1 through 15, and tt represents a transmitter unit number in the range of 0 through 15. Thus, four CPU's, each running foreground and background programs, may have ten possible lines connections, as shown in Figure 8.1.

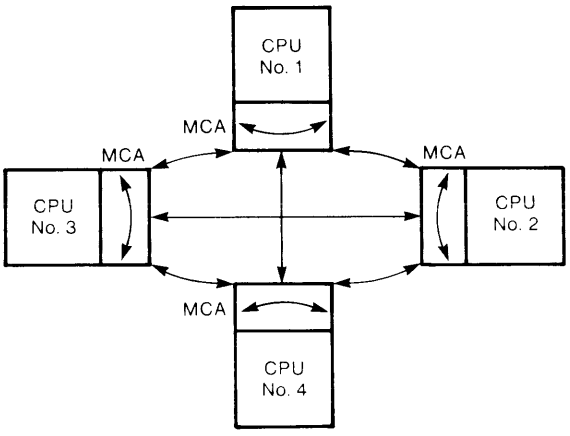


Figure 8.1 Multiple processor line connections SD-00554

Referring to Figure 8.1 and assuming that CPU 1 wants to read (receive) from CPU 3, each unit would issue the following sets of instructions:

```
CPU 1
.OPEN n          ;OPEN MCAR:3.
.RDS n           ;WAIT FOR THE DATA.
                 ;READ IT WHEN SENT.
CPU 3
.OPEN n          ;WRITE (TRANSMIT) TO.
.WRS n           ;WRITE (TRANSMIT) TO
                 ;THE RECEIVER LINE.
```

CPUs 1 and 3 operate under distinct RDOS systems. Thus, in the excerpt of code just shown, there is no relationship between channel n for unit 1 and channel n for unit 3.

A receiver can request a transmission from any transmitter by issuing a read call to receiver 0 (filename MCAR:0). After a receiver issues this call, any transmitter can write to it. Thus, if a program in CPU 1 had issued three receive requests—to MCAR:1, MCAR:3, and MCAR:0—it would receive transmissions from three sources: (1) from its own machine (transmission from the other ground), (2) from a program in CPU 3, and (3) from any other program that wanted to transmit to it. Each transmitter would transmit by issuing a write to MCAT:1.

All messages must begin on a word boundry, and the receive and transmit byte counts must match. To transmit an end of file, you can transmit a zero-byte message (eg, a .WRS operation of zero bytes).

A timeout occurs only in an MCA transmitter; a receiver can wait indefinitely. The timeout period ranges approximately from 200 miliseconds to 655 seconds. The default timeout is 655 seconds; you can select a shorter period when you open the MCA line and issue a write sequential. Refer to the descriptions of the .OPEN and .WRS commands in Chapter 3 for details.

Using CLI Commands on MCA Lines

As described earlier, each MCA line has a filename. This name can be used in conjunction with many of the CLI commands that take a filename in argument. The only special requirement is that the CLI command be present in both receiver and transmitter, since no data transmission can occur without simultaneous receive and transmit requests.

Moreover, you can transfer (via the XFER, but not the LOAD or DUMP, command) disk files across MCA lines. Thus, assuming the configuration shown earlier in Figure 8.1, the following CLI sessions would occur to transfer file ABC from CPU 4's disk to that of CPU 2.

From CPU 2, an operator enters this statement on the system console:

```
R
XFER MCAR:4 ABC ⟨CR⟩
```

With this command, CPU 2 tells its MCA to transfer the contents of MCAR:4 to ABC on its disk. Because CPU 2 is addressing a receive line, this is a receive request. Alternatively, the same operator could type:

```
R
XFER MCAR:0 ABC ⟨CR⟩
```

With this statement, CPU 2 tells its MCA to transfer any transmitter's input to ABC on its disk.

From CPU 4, an operator enters the following command line on the system console:

```
R
XFER ABC MCAT:2 ⟨CR⟩
```

With this statement, CPU 4 tells its MCA to transfer the contents of ABC on its disk to MCAT:2. Because CPU 4 is addressing its transmitter, this is a transmit request.

When a CPU issues a CLI command over an MCA line, the CLI prompt does not return to its console until RDOS has executed the command—or, if the transmitter issues the request—until the transmitter has timed out.

Transmitting Copies of Systems or Stand-alone Programs

RDOS provides a bootstrap program, MCABOOT, that transfers and bootstraps a copy of an RDOS system to another unit's disk. Before sending the system, MCABOOT can either fully or partially initialize the receiver's disk. Alternatively, MCABOOT can send and bootstrap a copy of a stand-alone program to another unit's disk, provided that this program follows the conventions of programs which BOOT can load. (BOOT need not reside in the receiving unit's disk space.) As with other MCA data transfers, both receiver and transmitter must participate.

You execute the MCA bootstrap by issuing the CLI's MCABOOT command. The transmitter and receiver must be on the same network (MCA or MCA1) for transmission to occur. An operator at the receiving CPU must have requested the transmission by placing 100007_8 (MCA) or 100047_8 (MCA1) in the receiver's data switches, and by pressing RESET followed by PROGRAM LOAD. The transmitting unit waits for the request from the receiver, but only up to the timeout period of 655 seconds.

.GMCA

Get the current CPU's MCA number

Your program can get the MCA unit number of its CPU by issuing system call .GMCA. It can then communicate this number to programs running in other CPUs.

Required Input

ACO - MCA transmitter octal device code (6 for MCAT, 46 for MCAT1).

Format

```
.SYSTM
.GMCA
error return
normal return
```

Upon the normal return, AC1 contains the MCA unit number.

Possible Errors

AC2	Mnemonic	Meaning
3	ERICD	Improper device code input to system call
36	ERDNM	Device not in system, that is, you did not specify an MCA for this RDOS system during system generation.

Multiprocessor System Illustration

This section illustrates one application of a multiprocessor system. It assumes a large, laboratory complex in need of an automated system to control the environmental conditions within the complex; to keep track of the number of personnel at different locations; to monitor the complex for alarm conditions; and to alert key personnel if it cannot correct a condition. This system must be fail-safe, and can allow down-time for no longer than a few seconds.

Figure 8.2 suggests one configuration for this system. Two master CPUs, running under mapped RDOS, are connected via an IPB, so that each can act as a watchdog on the other's behavior and can take control if the other fails. The IPB also allows the CPUs to access common disk files. The masters access a common data base which contains, among other information, alarm messages and destinations to which they should go on an alert. This file space also contains a log of the current master's activity, providing a record of recent events for the alternate, master CPU in the event that the current one fails.

The laboratory includes three vital zones; a slave CPU monitors and controls conditons within each zone. Each slave can monitor and adjust both humidity and temperature. Additionally, each slave keeps track of the positions of personnel within each zone. Finally, each slave monitors its zone for alarm conditions; if they occur, it takes some remedial action, such as, activating a sprinkler system in the event of fire. Each slave computer performs relatively simple operations, and could therefore run under RTOS, a core-resident compatible subset of RDOS.

Each slave has a data channel line through its MCA to each master computer (lines MCA1 through MCA6). This allows the current master to generate continuous status reports and transmit them to CRT monitors via the bus switch to an ALM. An SLM multiplexor connects "hot lines" to security guards and fire station personnel to alert them in an emergency.

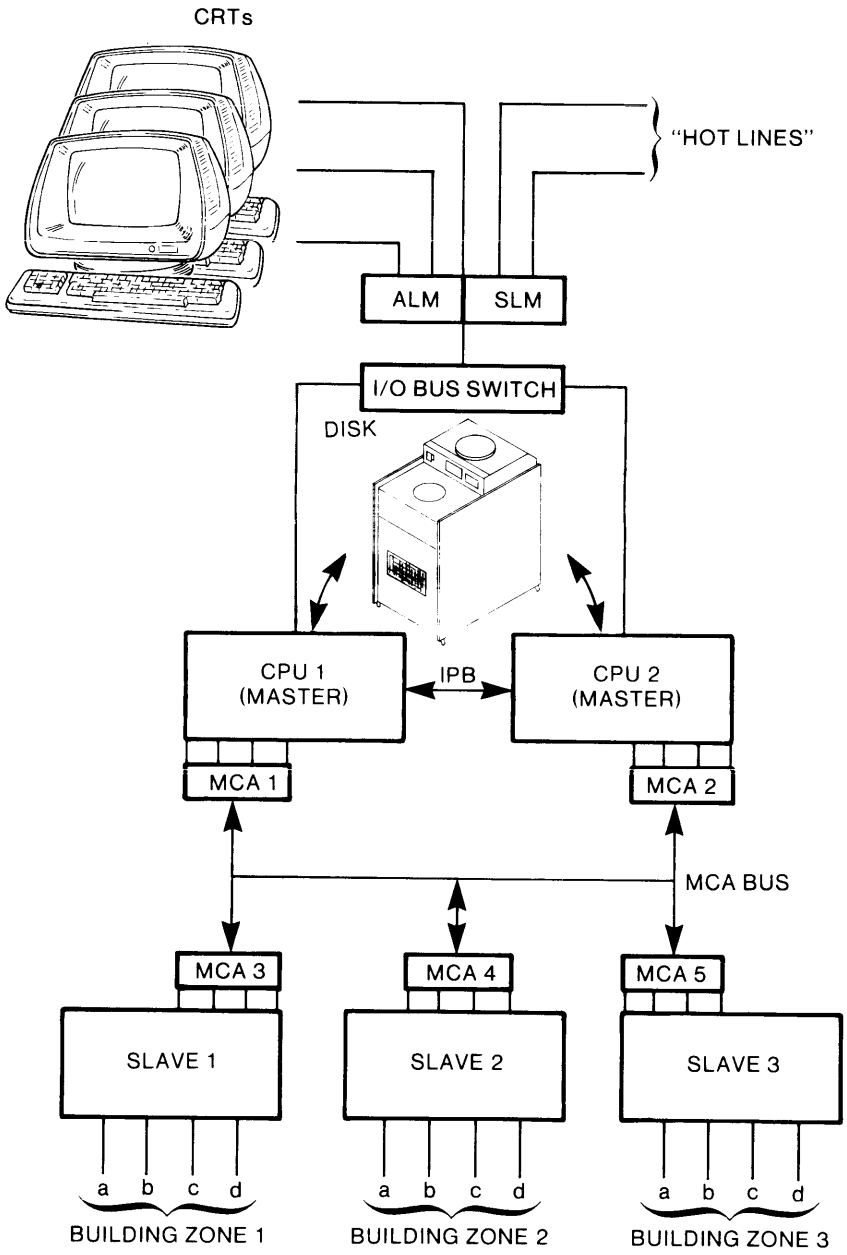


Figure 8.2 Multiprocessor system illustration

DG-25449

Chapter 9

System Tuning

This chapter describes the tuning facility, which allows an RDOS system to monitor its own performance and suggests more efficient configurations for any application. The chapter begins with a discussion of the data structures involved in tuning, including system stacks, cells, and buffers. In the course of this discussion, the RDOS system overlays are listed by name and function. Then the operation of tuning is explained, followed by a description of pertinent system calls.

Overview

During system generation, you tailor an RDOS system for a specific environment by answering the questions of the program SYSGEN.SV. (For details on SYSGEN, refer to *How to Load and Generate RDOS*, DGC No. 069-400013. This manual also provides a practical discussion on tuning.) Your answers to SYSGEN's questions determine the features that your RDOS system will include, and the peripheral hardware that it will support. SYSGEN also asks questions about the tuning facility, allowing you to choose whether your RDOS sytem will have tuning at all, and how extensive the tuning function will be.

The tuning mechanism itself deals with certain software data structures, called *stacks*, *cells*, and *buffers*. SYSGEN asks you to supply specifications for each of these structures; the tuning mechanism takes your answers and tests them as RDOS runs. It can then print a tuning report that allows you to decide on more efficient answers, or it can instruct SYSGEN to modify your original answers when you generate a new system.

The latter approach, called *self-tuning* can generate a moderately efficient version of RDOS for any application. During self-tuning, SYSGEN examines a previously generated tuning report file and selects more appropriate responses to questions about buffers, stacks, and cells. You can direct a system to tune itself by including the name of the original SYSGEN dialog file, along with the /T switch, in the command line that invokes SYSGEN:

```
SYSGEN dialog-file/A tuning-file/T ⟨CR⟩
```

SYSGEN examines the tuning file and attempts to generate a system more efficient for this application than the one that was running when the tuning file was recorded. During self-tuning, SYSGEN does not have a global view: it has only the tuning file to work from, and must therefore make certain, arbitrary decisions—the value of user memory to this given application, for example. As a result, SYSGEN cannot completely determine the impact of tuning decisions upon any given application's efficiency. Nonetheless, it does an adequate job for applications not requiring maximum efficiency. By themselves, tuning file statistics are helpful; but, in the final analysis, comparative timing of different system configurations provides the true measure of efficiency.

System Stacks, Cells, and Buffers

Before exploring tuning, it is important to understand how system buffers, stacks, and cells are defined. RDOS is partially core-resident and partially disk-resident. This design enables RDOS to offer features ordinarily found only on larger operating systems, while the total, memory-resident portion of the system remains modest. Stacks, cells, and system buffers are all memory-resident parts of RDOS.

RDOS uses a system stack as a data base, to execute each concurrent system call. The greater the number of outstanding .SYSTM requests, the more system stacks RDOS needs to service each request in parallel. If, for example, two executing user tasks issue a system call concurrently, two system tasks are then outstanding. To service both system tasks in parallel, RDOS would require two system stacks. At a single moment, RDOS services only as many requests as it has available system stacks, in the order that these calls were made. System tasks are associated not only with system calls, but also with I/O device requests and with spooling.

Each system task also requires a cell, to save state information, just as each user task has a task control block. There is a fundamental difference between cells and TCBs, however: RDOS sometimes appropriates cells for temporary data storage, but it never uses TCBs for this purpose.

A large part of memory-resident RDOS is a collection of system buffers, which serve two functions. First, RDOS uses buffers to receive system overlays, which provide code not found in the resident portion of the system. Second, RDOS buffers all I/O, except read/write block operations,

via system buffers. RDOS requests and uses system stacks, buffers and cells dynamically, as resources. When it needs and cannot get any of these resources a fault occurs, it suspends the calling system task, and system operation suffers.

System Stack Requirements

The following guidelines will help you select the proper number of system stacks during system generation. RDOS requires stacks for disk I/O, spooling, and the concurrent execution of system calls, as follows:

System Task	Number of stacks required
Disk I/O	Two stacks if you will be running multitask programs, or foreground and background programs that need to issue disk I/O system calls concurrently (eg, .OPEN, .INIT, .WRL).
Spooling	One stack.
System calls	One stack for each user task permitted to execute a .SYSTM call (requiring the use of an I/O device) concurrently with other user tasks.

SYSGEN offers a choice of one to 10 (decimal) system stacks. If this RDOS system will run single-task programs in a background-only environment, you need only specify one system stack. To spool output data, add another system stack. If you allocate only one stack, RDOS will not spool and will treat any system spooling commands that you issue as no-ops. Likewise, if a system also has a foreground program active and you have defined only two stacks, no spooling will occur. At least two stacks are recommended for a single-ground system; three, for a dual-ground system; and more for Extended BASIC.

To illustrate further, suppose that you plan a background-only, multitask program that spools to the line printer and performs disk I/O on only one channel at a time. This program requires the allocation of three system stacks: one stack for disk I/O, a second for line printer output, and a third for the spooler.

In general, you should allocate enough stacks to prevent system calls issued to slow peripherals ($PTR, MTA, etc.) from interfering with system calls necessary to support a real-time environment. Each .RDL or .RDS call to a non-multiplexed console requires a system stack until the read is completed.

Each system stack requires approximately 250_8 or 350_8 words, depending on your computer; the manual *How to Load and Generate RDOS* provides exact figures. Add the stack total to the basic memory requirements of the RDOS system.

When the system attempts to allocate a stack and none is free, it suspends the calling task and passes control to the next system task that is ready for execution; RDOS will attempt to allocate a stack for the suspended task at some future moment. Thus, the tuning report may indicate multiple, unsuccessful stack requests for the same system task. The same is true of certain cell requests. However, all unsuccessful buffer and overlay requests, and most unsuccessful cell requests, cause the system task to wait until the appropriate resource becomes free.

System Cell Requirements

A system cell is a 20_8-word control table that the system uses primarily to save system task state information. The optimum number of cells depends largely upon your system's application

.SYSGEN automatically allocates two cells for future read/write block operations; three cells for each stack; and two cells for an IPB, if you selected this option. It is recommended that you specify two extra cells for each active spool request, and one or two extra cells to improve the performance of the IPB, if any. Each active system call also needs an extra cell.

Since one goal of tuning is to keep all peripheral devices active concurrently,you need not allocate a cell for every possible, future concurrent system call. For slow peripherals, a lack of cells can degrade the system's operation. Consider the apportionments of cells illustrated in Figures 9.1 and 9.2. In Figure 9.1, this RDOS system contains three devices—a disk, a tape drive, and a line printer—and has nine cells. The program environment contains 20 user tasks, each one desiring the use of each of the three devices. As it happens, these tasks want to use different devices; hence, the system runs efficiently. As each task issues an I/O request, RDOS enqueues its cell to that device so that the next task in line will eventually be able to use it when it becomes free. Thus, RDOS enqueues only nine system tasks for the devices (and stores 11 requests in a special system table, PTBL). Even though 11 requests are waiting in table PTBL, the system runs efficiently.

Notice the difference in Figure 9.2. This is the same system, except that nine ready tasks want to use the magnetic tape drive; and these tasks monopolize the cell queue. Although up to 11 other tasks want to use the disk and line printer, they can not be readied until they receive a cell. RDOS frees cells one by one as the ready tasks finish with the tape drives; meanwhile, the other tasks stagnate in PTBL. Nine cells are too few for this program, although this number is sufficient for the same system and a different program, demonstrated in Figure 9.1. The waiting tasks cannot use

the disk and line printer, even though these devices are not busy, and the system is running inefficiently.

RDOS would report cell faults in *both* of the environments illustrated by these figures. Yet additional cells would not improve system efficiency in the example of Figure 9.1. Thus, you must supplement the fault information provided in the tuning report by timing your application programs to determine whether the reported faults actually degrade system performance.

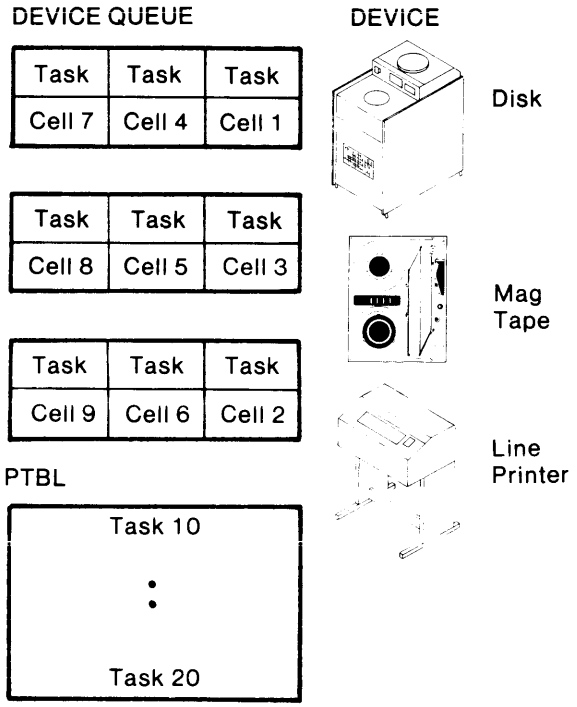


Figure 9.1 Adequate cell apportionment DG-25450

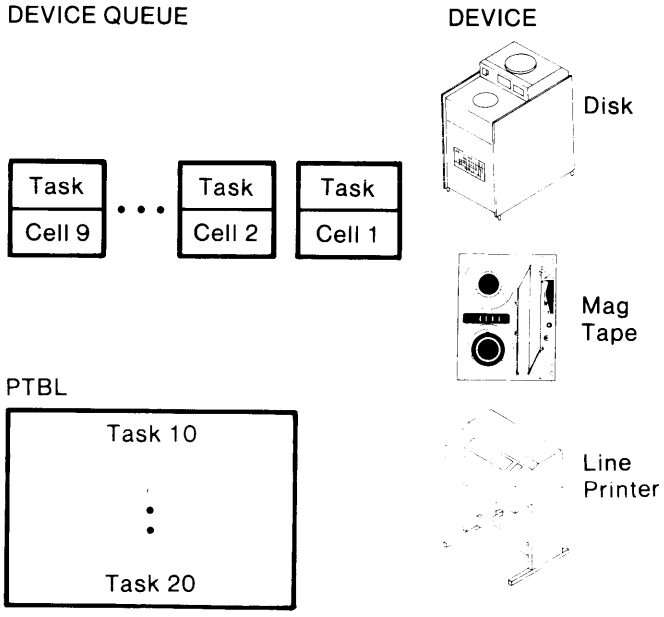


Figure 9.2 Inadequate cell apportionment DG-25451

System Buffer Requirements

System buffers are portions of memory which RDOS allocates dynamically to receive either user data or system overlays. RDOS requires a minimum of two buffers per system stack, or six buffers total, whichever is greater. SYSGEN automatically allocates this minimum; during system generation, you can specify as many extra buffers as core memory will allow. Each system buffer requires 416_8 or 274_{10} words. In mapped systems, any multiples of 274 words available in the last 1024-word block of system space are used by RDOS for additional system buffers.

When RDOS needs a buffer, it flushes the contents of the oldest buffer that is not in use. However, if extra buffers are available, fewer of them are flushed and their contents remain accessible to system memory. If your application favors having buffered data in core (for fast reaccess), or having many system overlays resident in core (for fast system call execution), you should specify extra buffers. Extra buffers increase system speed but reduce the total amount of memory available for your programs. The ideal solution incorporates enough system buffers to provide the desired speed while leaving adequate memory for your programs.

RDOS requires some system buffers to receive system overlays. Table 9.1 describes each overlay and the system calls or functions that it executes. Each overlay's number (octal) precedes its name in the list; you will need this number to understand the tuning report, since the report does not refer to system overlays by their names.

Overlay Number	Name	Functions
0	DFRWS	Disk file .WRL, .RDR/.WRR, .RDS/.WRS.
1	DFRWS	Disk file .CHSTS, .RDL, .LINK, .RDL, .STAT.
2	UTIL1	Magnetic tape .GCHN, .GMEM, .SMEM; tape .MTDIO.
3	CREATE	Starts file creation: .CONN, .CCONT, .CRAND, .CREAT.
4	DELETE	Delete a file, a subdirectory, or a secondary partition: .DELET.
5	FILSY	Maintains directories and searches for entries in them.
6	SOV1	Implements periodic rescheduling for .QTASK and QUE. Also implements the following system calls: .CHATR, .CHLAT, .FGND, .GCIN, .GCOUT, .GTATR, .GTOD, .ODIS, .OEBL, and .STOD.
7	SOV2	Checks filenames for validity, interprets directory specifier prefixes, and unpacks file names into SYS.DR format.
11	SOV3	Processes disk file errors and reads disk core images.
11	SOV4	Opens files (.OPEN, .EOPEN, .ROPEN); .CLOSE, .RESET, and implements CLI CLEAR command.
12	DVINI	Initializes directories; .EQIV.
13	CRSFS	Creates a MAP.DR entry in SYS.DR, and creates peripheral device entries in SYS.DR after a full initialization.
14	RING1	Opens and closes character devices on level; writes messages to the console.
15	RING2	Console keyboard and reader .RDL/.RDS; .RDS, .GCHAR: system-level character I/O (ACHR, WRS, PCH).
16	RING3	Performs system-level character I/O (ACHR, WRS, PCH).
17	SOV5	Performs housekeeping necessary to execute keyboard interrupt or .BREAK.
20	MTAIO	.INIT, .RLSE, .CLOSE; block-level reading and writing for magnetic tapes units.

Table 9.1 System overlays and their functions

Overlay Number	Name	Functions
21	MTAUC	.OPEN for magnetic tape units.
22	TUON	.TUON: turn tuning on.
23	CDROV	Card reader ASCII .RDL.
24	WDBLK	Completes the file creation activities originating in CREATE; withdraws a single block from MAP.DR.
25	SPOLR	Supports spooling.
26	CODER	Encodes and decodes 7-track magnetic tape; .SKPK, .SPDA, .SPEA.
27	SOV6	Writes a core image to disk.
30	SOV7	Continues disk core image read function started by SOV3.
31	SOV8	.EXEC, .EXFG, .EXBG.
32	SOV9	Resolves directory link entries; .EXEC, .EXFG, .EXBG.
33	SOV10	Continues directory resolution function of SOV9; .INIT, .RLSE.
34	SOV11	Determines size of a fixed-head disk for .INIT system call; .ICMN, .RDCM, .WRCM.
35	JEHOV	Creates an initial system directory.
36	SOV12	Continues the function performed by SOV5; creates file BREAK.SV and completes a program break caused either by . BREAK or console keyboard interrupt.
37	SOV13	Opens, closes a disk file; .UPDAT.
40	SOV14	.DIR, .RDOP/.WROP.
41	SOV15	.CDIR, .CPART.
42	SOV16	.IDEF, .DEBL/.DDIS.
43	FILS2	Preprocesses the deletion of partitions and subdirectories; .RENAM.
44	SOV17	Finishes the housekeeping started by SOV5 for .EXEC/.RTN and keyboard interrupts; .IRMV.

Table 9.1 System overlays and their functions (continued)

Overlay Number	Name	Functions
45	SOV18	Produces an orderly shutdown upon a system release; .BOOT.
46	WDCBK	Withdraws a series of contiguous blocks from MAP.DR; creates an elemental MAP.DR for DIVINI and SOV15.
47	SOV19	Determines size of a moving head disk during .INIT system call; performs QTY open/close.
50	SOV20	Prepares program environment for a core-image load (mapped systems only).
51	SOV21	Provides MCA read/write sequential and other MCA support functions.
52	SFTAB	Data overlay used to build pd peripheral device entries during a full system initialization.
53	SOV22	Continues the code begun in SOV18; .OVRP.
54	SOV23	Aborts a system process.
55	SOV24	Resolves spooling deadlocks; .GPOS/.SPOS, .CA; .OPEN for MCA.
56	SOV25	Completes the operation initiated by overlay 46 (WDCBK).
57	FSTAT	Provides support to other system overlays by getting and/or updating file status and by obtaining block address for disk I/O. Deposits a free block in MAP.DR.
60	DVRLS	Releases a directory; determines the DCT of a device for the spooling routines in overlay 26 (CODER); .GDIR, .MDIR, .GSYS.
61	SOV26	.WRPR, .WREBL, .STMAP.
62	SOV27	Completes the execute functions started in SOV8; continues the functions performed by SOV3, SOV5, and SOV12; .RTN/.ERTN.
63	SOV28	.OVOPN; .MAPDF and .VMEM for mapped systems only.
64	TUNOV	.TUOFF: turn tuning off.
65	QTYOV	Provides QTY/ALM driver support.
66	SOV29	Replaces overlays in an overlay file.

Table 9.1 System overlays and their functions (continued)

How Tuning Works

After you have generated a system with tuning (having specified a number of stacks, cells, and system buffers), you can turn tuning on and start recording in the tuning file. If you find your system inefficient, you can examine the tuning report and generate a new system, specifying a different number of stacks, cells, and/or buffers. As mentioned earlier, you can also instruct SYSGEN to examine the tuning file and modify your original answers to these questions. This procedure, known as self-tuning, can be performed as often as needed to arrive at one or more RDOS systems that run your application(s) well. You will cause a system failure, however, if you activate tuning in a system before deleting the tuning file of a previous system with the same name.

As with many RDOS features, you can use either system calls or CLI commands to turn tuning on or off. You must, of course, have selected the tuning option during system generation, along with the type of tuning report you desire. SYSGEN automatically reserves extra buffers within the system for use by the report function. One buffer is required for the summary report; detailed reports require three buffers.

The CLI commands that turn tuning on and off are TUON and TUOFF, respectively. The command that displays the contents of the tuning file is TPRINT. The system calls corresponding to these commands are .TUON and .TUOFF. Use of the CLEAR command to clear the tuning file does not turn tuning off or affect the report file. However, you must not delete this file while tuning is on. To produce a fresh tuning report, issue CLI commands TUOFF, RENAME or DELETE, and TUON.

When tuning is on, the tuning feature accumulates the number of requests for stacks, cells, buffers, and system overlays. RDOS records this information in a disk file named sysname.TU, where sysname is the name of the current RDOS system. This file resides in the master directory. Additionally, RDOS records the number of times it defaulted a request because the resource was not available. You can then compute the ratio of requests to faults as an indication of your system's efficiency.

Note that your program can access the tuning file by opening it and then issuing system call .RDS for 2*TULEN bytes. (TULEN defines the number of words in the summary report).

The tuning report file is a contiguous disk file consisting of either one or three disk blocks, depending on whether you requested an overlay report during system generation. The first disk block contains the summary report. The overlay report, if requested, follows on the next two disk blocks. Figure 9.3 shows the composition of disk blocks in the tuning file.

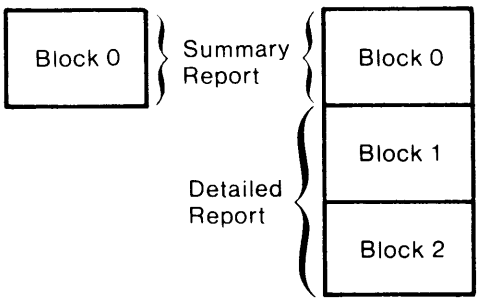


Figure 9.3 Disk blocks of the tuning file SD-00569

The summary report contains four sections: one each for system stacks, cells, buffers, and overlays. Each section in the summary is composed of five, 16-bit words. The first word in each section lists the number of elements (stacks, buffers, etc.) in the system. The next two words are a double-precision integer count (two, 16-bit words) of all requests for this element. The last two words are a double-precision integer count of faults, ie, unsuccessful requests for the resource. Each double-precision count returns to zero upon overflow. The remaining words in the summary disk block are not meaningful.

Figure 9.4 shows the arrangement of information in the summary portion of the tuning report file. The named word displacements relative to the beginning of the file are defined in PARU.SR, a file of user parameters supplied with your RDOS system and listed in Appendix B.

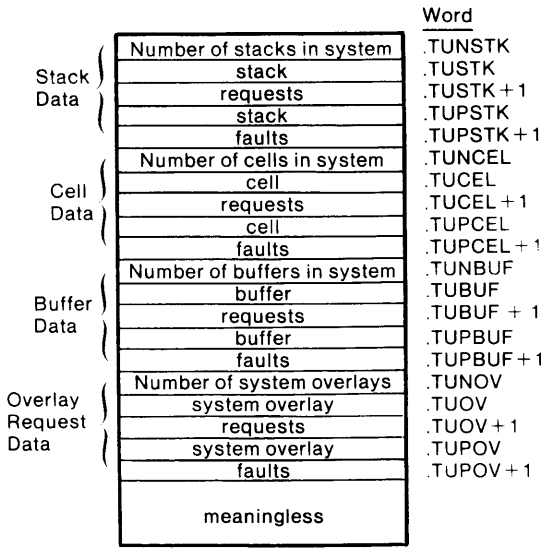


Figure 9.4 Details of the tuning summary report, first disk block SD-00570

Referring to Figure 9.4, the number of stacks and cells (displacements .TUNSTK and .TUNCEL) is the total of each in the system; the number of buffers (displacement .TUNBUF) is the total number of buffers, excluding tuning buffers. The buffer request count reflects requests for buffers needed to receive data or system overlays. As indicated earlier, multiple stack and cell faults can occur and be recorded for the same system task.

If you specified a detailed tuning report during system generation, RDOS places it in the blocks immediately following the summary in the tuning report file. The detailed report consists of a series of four-word descriptors, with one descriptor for each system overlay. Each descriptor contains a count of requests for a system overlay, and a count of the number of requests that required the overlay to be read from disk because it was not then resident in memory. Each count is a double-precision integer; if an overflow occurs, RDOS returns the count to zero. The detailed report can list up to 128, separate system overlays. The counts of defined, but unused, overlays are set to zero.

Figure 9.5 depicts the arrangement of information in the detailed tuning report for a system with *m* overlays. Each system overlay was described earlier in Table 9.1.

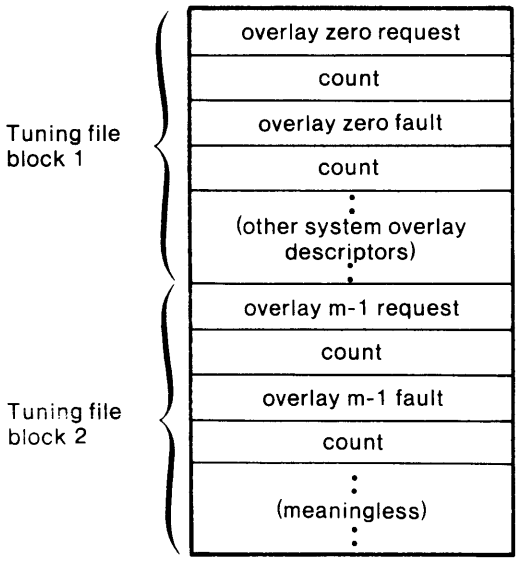


Figure 9.5 Tuning overlay report SD-00571

.TUON

Start recording in the tuning file

This system call turns on the tuning mechanism, which reports system resources and faults in the tuning file. If the tuning report file does not exist, this command creates it as a contiguous file of either one or three blocks; the size depends upon your choice of report functions during system generation. RDOS names the file sysname.TU, where sysname is the name of the current RDOS system. This file resides in the master directory.

If the tuning file already exists, any new information will be added to it. If the tuning report function is already on, the .TUON command is an effective no-op.

Required Input

AC0 - Set to zero.

Format

```
.SYSTM
.TUON
error return
normal return
```

Possible Errors

AC2	Mnemonic	Meaning
2	ERICM	Illegal system command. (Tuning was not selected during system generation.)
27	ERSPC	Insufficient disk space to create tuning file.
46	ERICB	Insufficient number of free contiguous disk blocks available to create the tuning file.
101	ERDTO	Disk timeout occurred.

.TUOFF

Stop recording in the tuning file

This system call halts the tuning report function until and unless you turn it back on with the .TUON command. The .TUOFF command does not delete the tuning file itself. Any extra system buffers used by the tuning function are released to the system. If the tuning report function is already turned off, this call is an effective no-op.

Required Input

None.

Format

```
.SYSTM
.TUOFF
error return
normal return
```

Possible Errors

AC2	Mnemonic	Meaning
12	ERDLE	Tuning file was deleted before tuning was turned off.
101	ERDTO	Disk timeout occurred.

Chapter 10

Running In LEF Mode

This chapter applies to users running RDOS on mapped, ECLIPSE computers. It describes the Load Effective Address (LEF) instruction, which allows you to load an address directly into an accumulator, or to load, add, or subtract a constant between +127 and −128 (177_8 and -200_8) to an accumulator without using a separate memory location to hold that constant.

Before you issue this instruction, you must set bit 9 in the user status word to put the CPU into LEF mode. Task call .LEFE, described in this chapter, serves this purpose. Once the CPU is in LEF mode, you cannot issue I/O instructions because the system interprets them as LEF instructions. To disable LEF mode, use task call .LEFD.

When you plan to use LEF mode, you can determine whether it is currently set or reset with task call .LEFS. This and other LEF calls are used according to your program's needs. If, for example, you define a device service routine via the .IDEF command, you must start the device at base or program level in order for it to generate an interrupt and be serviced. If the CPU is in LEF mode, you cannot issue the device start (or any other I/O) instruction until the LEF mode is disabled. Further details on LEF mode and the LEF instruction can be found in *Programmer's Reference Manual, ECLIPSE Line Computers*, 014-000626.

By default, LEF mode is disabled. Once you enable it, it remains set only for the duration of current program; a .RTN or .EXEC call will disable it for the new program. The LEF tasks calls are:

.LEFD	Disable the LEF mode.
.LEFE	Enable the LEF mode.
.LEFS	Get the LEF mode status.

As with task calls you must reference these names in a .EXTN statement before issuing the calls. LEF commands may be issued in both single- and multitask environments.

.LEFD

Disable the LEF mode

This task call disables the LEF mode. After you issue it, single-word LEF instructions cannot be issued. If the CPU is currently set with LEF mode disabled, this call becomes an effective no-op. The contents of AC0 and AC3 are lost upon return.

Required Input

None.

Format

```
.LEFD
normal return
```

Possible Errors

None.

.LEFE

Enable the LEF mode

This task call enables the LEF mode and allows user programs to issue single-word LEF instructions. If the LEF mode is already set on your CPU, this call becomes a no-op. The contents of AC0 and AC3 are lost upon return from this command.

Required Input

None.

Format

```
.LEFE
normal return
```

Possible Errors

None.

.LEFS

Get the LEF mode status

Issue this task call to determine whether the LEF mode is currently set or reset in the CPU. When you issue this call, RDOS returns the user status word in AC0; bit 9 of this word is set only if you enabled LEF mode. Consult the *Programmer's Reference Manual, ECLIPSE Line Computers* (DGC No. 014-00626) for a complete definition of the user status word. The contents of AC3 are lost upon return.

Required Input

None.

Format

```
.LEFS
normal return
```

AC0 contains the user status word upon return.

Possible Errors

None.

Appendices

The nine appendices that follow supplement the foregoing chapters with command and error summaries, programming examples, conversion tables, and more. In order of appearance, the appendices include:

- Appendix A: RDOS System and Task Calls and Error Summary
- Appendix B: User Parameters
- Appendix C: Real-time Programming Examples
- Appendix D: Overlay Directory Structure
- Appendix E: Exceptional System Status
- Appendix F: Page Zero and Hardware Reserved Locations
- Appendix G: Hollerith-ASCII Conversion Table
- Appendix H: ASCII Characater Set
- Appendix I: Advanced Multitask Programming

Appendix A

RDOS System and Task Calls and Error Summary

Table A.1 describes each RDOS system and task call, along with the required input to (or remarks on) the accumulators. Variable n in this table represents the file's channel number, as assigned on the open. After a task or system call, AC3 contains the user stack pointer (USP) by default. To return the frame pointer, refer to the section "System and Task Calls" in Chapter 3. RDOS returns error codes, if any, in AC2. Task calls sometime destroy accumulators, as noted. System calls preserve accumulators if they do not specifically return values.

Table A.2 lists and describes error codes in numeric order.

Command	Description	Input and Remarks
.ABORT	Abort a task.	AC0: Destroyed. AC1: Task ID number in bits 8-15.
.AKILL[1]	Kill all tasks of a given priority.	AC0: Priority of task to be killed.
.SYSTM .APPEND n	Open a file for appending.	AC0: Byte pointer to filename. AC1: Device characteristic mask (see .GTATR). AC2: Channel number, if n = 77.
.ARDY[1]	Ready all tasks of a given priority.	AC0: Priority of tasks to be readied.
.ASUSP[1]	Suspend all tasks of a given priority.	AC0: Priority of tasks to be suspended.
.SYSTM .BOOT[2]	Bootstrap a new system.	AC0: Byte pointer to *primary partition* or *specifier: filename*.
.SYSTM[1] .BREAK[2]	Interrupt the current program and save the current state of memory in save file format.	
.SYSTM .CCONT	Create a contiguously organized file with all data words zeroed.	AC0: Byte pointer to filename. AC1: Integer number of disk blocks.
.SYSTM .CDIR	Create a subdirectory.	AC0: Byte directory to new directory name.

Table A.1 RDOS command summary

Command	Description	Input and Remarks
.SYSTM .CHATR n	Change file attributes.	AC0: 1B0, read-protect this file. 1B1, attribute-protect this file. 1B7, allow no link resolution. 1B9, user attribute. 1B10, user attribute. 1B14, make this file permanent. 1B15, write-protect this file. AC2: Channel number, if n=77.
.SYSTM .CHLAT n	Change link access attributes.	AC0: Same as .CHATR, above. AC2: Channel number, of n=77.
.SYSTM .CHSTS n	Get the status of a file currently open on a specified channel.	AC0: Starting address of 22_8-word area. AC2: Channel number, if n=77.
.SYSTM .CLOSE n	Close a file.	AC2: Channel number, if n=77.
.SYSTM .CONN	Create a contiguously organized file with no data words zeroed.	AC0: Byte pointer to filename. AC1: Integer number of disk blocks.
.SYSTM .CRAND	Create a random file.	AC0: Byte pointer to filename.
.SYSTM .CPART	Create a secondary partition.	AC0: Byte pointer to secondary partition name. AC1: Number of contiguous blocks (must exceed 60_8).
.SYSTM .CREAT	Create a sequential file.	AC0: Byte pointer to filename.
.SYSTM .DDIS	Disable user access to a device in a mapped system.	AC0: Device code to be disabled from user access.
.SYSTM .DEBL	Enable user accesss to a device in a mapped system.	AC0: Device code to be enabled for user access.
.SYSTM .DELAY	Delay the execution of a task.	AC1: Number of RTC pulses.
.SYSTM .DELET	Delete a file.	AC0: Byte pointer to filename.
.SYSTM .DIR	Change the current directory.	AC0: Byte pointer to directory/directory device specifier.
.DQTSK	Dequeue a previously queued task.	AC1: Task ID number in bits 8-15. AC2: Base address (returned) of released queue area.
.DRSCH[1]	Disable the task scheduler.	

Table A.1 RDOS command summary (continued)

Command	Description	Input and Remarks
.SYSTM .DUCLK	Define a user clock.	AC0: Number of RTC pulses. AC1: Address of user interrupt routine.
.SYSTM .EOPEN n	Open a file for reading and writing by one user only.	AC0: Byte pointer to filename. AC1: Characteristic disable mask (see .GTATR). 0 leaves characteristics unchanged. AC2: Channel number, if n = 77.
.SYSTM .EQIV	Assign a temporary name to a device.	AC0: Byte pointer to current disk or tape specifier. AC1: Byte pointer to temporary specifier.
.SYSTM .ERDB	Read one or more disk blocks into extended, mapped memory.	AC0: External memory block number (0,1,2, or 3) in right byte. AC1: Starting relative block number in disk file. AC2: In right byte, number of 256-word blocks to be read. In left byte, channel number if n = 77.[3]
.ERSCH[1]	Reenable the task scheduler.	
.SYSTM .ERTN[2]	On an error, return from program and describe error (if to CLI).	AC2: Data word to be passed to next-higher level.
.SYSTM .EWRB	Write one or more 256-word blocks from extended, mapped memory to disk.	AC0: Extended memory block number in right byte. 256-word group number (0,1,2, or 3) in left byte. AC1: Starting relative block number in disk file. AC2: In right byte, number of 256-word blocks to be written. In left byte, channel number if n = 77.[3]
.SYSTM .EXBG	Checkpoint a background program in a mapped system.	AC0: Byte pointer to new background program's name. AC1: 0B1, new background to have same priority as old. 1B1, new background to have same priority as foreground. AC2: Optional message to new background.
.SYSTM .EXEC	Swap or chain in a new program.	AC0: Byte pointer to new save file's name. AC1: 0, swap to user program. 1B0, chain to user program. 1, swap to debugger. 1B0 + 1, chain to debugger.

Table A.1 RDOS command summary (continued)

Command	Description	Input and Remarks
.SYSTM .EXFG	Execute a program in the foreground.	AC0: Byte pointer to save file's name. AC1: 0B1, foreground to have over background. 1B1, foreground/background equal priority. 0B15, pass control to save file. 1B15, pass control to debugger.
.SYSTM .FGND	Determine whether or not a foreground program is running.	AC0: (returned) 0. AC1: (returned) program level code; 1 = background level 0 ... 12 = foreground level 4.
.SYSTM .GCHAR	Get character from the console.	AC0: Bits 0-8 cleared; character returned in bits 9-15.
.SYSTM .GCHN	Get the number of a free channel.	AC2: (returned) free channel number.
.SYSTM .GCIN	Get the operator input console name.	AC0: Byte pointer to 6-byte area receiving the input console's name.
.SYSTM .GCOUT	Get the operator output console name.	AC0: Byte pointer to 6-byte area receiving the output console's name.
.SYSTM .GDAY	Get today's date.	AC0: (returned) day. AC1: (returned) month. AC2: (returned) year minus 1968.
.SYSTM .GDIR	Get the current directory name.	AC0: Byte pointer to 13_8-byte area.
.SYSTM .GHRZ	Examine the real time clock.	AC0: (returned) 0 = no RTC 1 = 10 HZ 2 = 100 HZ 3 = 1000 HZ 4 = 60 HZ 5 = 50 HZ
.SYSTM .GMCA	Get the current MCA unit number.	AC0: MCA transmitter device code (6 or 46 octal). AC1: (returned) MCA unit number.
.SYSTM .GPOS n	Get the current file pointer.	AC0: (returned) high-order portion of byte pointer. AC1: (returned) low-order portion of byte pointer. AC2: Channel number, if n = 77.
.SYSTM .GSYS	Get the name of the current operating system.	AC0: Byte pointer to 15_8-byte area.

Table A.1 RDOS command summary (continued)

Command	Description	Input and Remarks
.SYSTM .GTATR n	Get file attributes.	AC0: (returned) 1B0, read-protected. 1B1, attribute-protected. 1B2, save file 1B3, link entry.[5] 1B4, partition.[5] 1B5, directory file.[5] 1B6, link resolution entry.[5] 1B7, no link resolution allowed. 1B9, user attribute. 1B10, user attribute. 1B12, contiguous file.[5] 1B13, random file.[5] 1B14, permanent file 1B15, write-protected AC1: (returned) MCA shares 0 and 15; see file PARU.SR 1B0, spoolable device. 1B1, 80-column card. 1B2, lower-to-uppercase. 1B3, form feed on open. 1B4, full word device. 1B6, LF after CR. 1B7, parity check/generation. 1B8, rubout after tab. 1B9, null after FF. 1B10, keyboard input. 1B11, TTY output. 1B12, no FF hardware. 1B13, operator intervention needed. 1B14, no TAB hardware. 1B15, leader/trailer. AC2: Channel number, if n = 77.

Table A.1 RDOS command summary (continued)

Command	Description	Input and Remarks
.SYSTM .GTOD	Get the time of day.	AC0: (returned) seconds. AC1: (returned) minutes. AC2: (returned) hours (using a 24-hour clock).
.SYSTM .ICMN	Define a program communications area.	AC0: Starting word address of communications area. AC1: Size of area in words.
.SYSTM .IDEF	Identify a user device.	AC0: Device code of user device. AC1: DCT. (1B0 if data channel device is mapped systems.) User power restart address if AC0 = 77_8.
.IDST[1]	Get a task's status	AC0: 0, ready. 1, suspended by .SYSTM call. 2, suspended by .SUSP, .TIDS, .AUSUSP. 3, waiting for .XMTW/.REC. 4, waiting for overlay node. 5, suspended by .SUSP, .ASUSP, or .TIDS and .SYSTM call. 6, suspended by .XMTW/.REC and .SUSP, .ASUSP, or .TIDS. 7, suspended by .ASUSP, .SUSP, or .TIDS and waiting for overlay node. 10, no such task exists. AC1: (returned) base address of task's TCB. AC2: Task ID in bits 8-15.
.SYSTM .INIT	Initialize a device or directory.	AC0: Byte pointer to directory/global device specifier. AC1: -1, full initialization (tape or disk). 0, partial initialization.
.SYSTM .INTAD	Define a program interrupt task.	
.SYSTM .IOPC	Initialize the Operator Communications Package (OPCOM).	AC0: Queue area address (0 if no RUN or QUE). AC1: In right byte, max number of queue areas (0 if no RUN or QUE). In left byte, overlay channel no. (0 if right byte = 0). AC2: Program table address (0 if no RUN or QUE).
.SYSTM .IRMV	Remove a user device.	AC0: Device code.

Table A.1 RDOS command summary (continued)

Command	Description	Input and Remarks
.IXMT	Transmit a one-word message from a user interrupt routine.	AC0: Message address (destroyed). AC1: Nonzero message (destroyed). AC2: Destroyed.
.KILAD[1]	Define a kill-processing address.	AC0: Address of kill-processing routine.
.KILL[1,2]	Kill the calling task.	
.LEFD[1]	Disable the LEF mode in a mapped, ECLIPSE system.	AC0: Contents lost upon return.
.LEFE[1]	Enable the LEF mode in a mapped, ECLIPSE system	AC0: Contents lost upon return.
.LEFS[1]	Get the LEF mode status in a mapped, ECLIPSE system.	AC0: (returned) user status word; 1B9, LEF mode is enabled. 0B9, LEF mode is disabled.
.SYSTM .LINK	Create a link entry.	AC0: Byte pointer to link name. AC1: If 0, link will be resolved in parent partition of link entry's residence. If not 0, byte pointer is either to an alternate directory alias name or to an alias name string.
.SYSTM .MAPDF	Define a window map in a mapped system.	AC0: Number of blocks for extended addressing use. AC1: Starting logical block number of window. AC2: Size of window in 1K blocks.
.SYSTM .MDIR	Get the logical name of the master device.	AC0: Byte pointer to 13_8-byte area.
.SYSTM .MEM	Determine available memory.	AC0: HMA. AC1: NMAX.
.SYSTM .MEMI	Change NMAX.	AC0: NMAX increment of decrement (2's complement). AC1: (returned) new NMAX (after change).

Table A.1 RDOS command summary (continued)

Command	Description	Input and Remarks
.SYSTM .MTDIO n	Perform free format I/O on tape or cassette.	AC0: Core data address, if a data transfer. Even parity if bit 0 = 1; odd parity if bit 0 = 0. Bits 1-3: 0, read (words). 1, rewind tape. 3, space forward. 4, space backwards. 5, write (words). 6, write EOF. 7, read device status word. Bits 4-15: word or record count; if 0 on space command, position tape to new file if it is less than 4096 records away. (returned) number of words read/written, or number of records spaced. AC1: Status word or system error code if error returns; status word if read status normal return. Returned: 1B0 error. 1B1, data late. 1B2, tape rewinding. 1B3, illegal command. 1B4, high density or cassette if 1; low density if 0. 1B5, parity error. 1B6, end of tape. 1B7, end of file. 1B8, tape at load point. 1B9, 9-track or cassette if 1; 7-track if 0. 1B10, bad tape; write failure. B11, send clock (0 if cassette). 1B12, first character (0 if cassette). 1B13, write-protected or write-locked. 1B14, odd character (0 if cassette). 1B15, unit ready.

Table A.1 RDOS command summary (continued)

Command	Description	Input and Remarks
.SYSTM .MTOPD n	Open a magnetic tape or cassette for free format I/O.	AC0: Byte pointer to tape global specifier. AC1: Characteristics inhibit mask (see .GTATR). AC2: Channel number, if n = 77.
.MULTI	Restore the multitask environment.	
.SYSTM .ODIS	Disable keyboard interrupts for this console.	
.SYSTM .OEBL	Enable keyboard interrupts for this console.	
.SYSTM .OPEN n	Open a file for reading and/or writing by one or more users.	AC0: Byte pointer to filename. AC1: Characateristic inhibit mask (see .GTATR). 0 leaves previous characteristic unchanged. 0 for MCA, or 1 to specify your own MCAT retry timeout. AC2: Channel number, if n = 77.
.OVEX[6]	Release an overlay and return to a specified address.	AC0: Node number in bits 0-7. Overlay number in bits 8-15. AC2: Return address.
.OVKIL[2]	Kill the calling task and release its overlay node.	AC0: Node number in bits 0-7. Overlay number in bits 8-15.
.SYSTM .OVLOD n	Load a user overlay into memory.	AC0: Node number in bits 0-7. Overlay number in bits 8-15. AC1: -1 for unconditional load; 0 for conditional load. AC2: Channel number, if n = 77.[4]
.SYSTM .OVOPN n	Open a user overlay file.	AC0: Byte pointer to overlay filename (with .OL extension). AC2: Channel number, if n = 77.[4]
.OVREL	Release an overlay.	AC0: Node number in bits 0-7. Overlay number in bits 8-15.
.SYSTM .OVRP	Replace an overlay file.	AC0: Byte pointer to overlay replacement filename. AC1: Byte pointer to overlay filename (with .OL extension).
.SYSTM .PCHAR	Write a character to the console.	AC0: Character in bits 9-15; bits 0-8 ignored.
.PRI[1]	Change a task's priority.	AC0: New task priority in bits 8-15. AC2: Address of user task queue table.

Table A.1 RDOS command summary (continued)

Command	Description	Input and Remarks
.QTSK	Queue a memory-resident or overlay task.	
.SYSTM .RDB n	Read one or more disk blocks.	AC0: Starting core address to receive data. AC1: Starting disk relative block number. AC2: Number of blocks to be read in bits 0-7; channel number, if n=77, in bits 8-15.[4]
.SYSTM .RDCMN	Read a message from the other program's communications area.	AC0: Word address to read into. AC1: Offset into communications area. AC2: Word count.
.SYSTM .RDL n	Read a line.	AC0: Byte pointer to user core area. AC1: (returned) read byte count, including terminator. AC2: Channel number, if n=77.
.SYSTM .RDOPR	Read an operator message.	AC0: Byte pointer to message area. AC1: (returned) byte count.
.SYSTM .RDR n	Read a random record.	AC0: Core address to receive record. AC1: Record number. AC2: Channel number, if n=77.
.SYSTM .RDS n	Read sequential bytes.	AC0: Byte pointer to core area (must be even for MCA). AC1: Number of bytes to be read. If EOF detected, partial byte count returned. AC2: Channel number, if n=77.
.SYSTM .RDSW	Read the console switches.	AC0: (returned) console with switch position.
.REC[1]	Receive a message from another task.	AC0: Message address. AC1: Message.
.REMAP	Activiate a logical window transfer in a mapped system.	AC0: Destroyed. AC1: Destroyed. In left byte, starting relative block number in map. In right byte, starting relative block number of window. AC2: Destroyed. Number of 1K blocks to be remapped.
.SYSTM .RENAM	Rename a file.	AC0: Byte pointer to old name. AC1: Byte pointer to new name.
.SYSTM .RESET	Close all files.	
.SYSTM .RLSE	Releae a directory or device.	AC0: Byte pointer to directory or global device specifier.

Table A.1 RDOS command summary (continued)

Command	Description	Input and Remarks
.SYSTM .ROPEN n	Open a file for reading only by one or more users.	AC0: Byte pointer to filename. AC1: Characteristic inhibit mask (see .GTATR). 0 preserves characteristics without change. For MCA, see .OPEN. AC2: Channel number, if n = 77.
.SYSTM .RSTAT	Get a resolution file's statistics.	AC0: Byte pointer to filename string. AC1: Starting address of 22_8-word area.
.SYSTM .RTN[1]	Return from a program to a higher-level program.	
.SYSTM .RUCLK	Remove a user clock.	
.SYSTM .SDAY	Set today's date.	AC0: Day. AC1: Month. AC2: Year minus 1968.
.SINGL	Disable the multitask environment.	
.SMSK[1]	Modify the current interrupt mask.	AC0: Lost. AC1: New interrupt mask to be ORed with old mask. AC2: Lost.
.SYSTM .SPDA	Disable spooling.	AC0: Byte pointer to device name.
.SYSTM .SPEA	Enable spooling.	AC0: Byte pointer to device name.
.SYSTM .SPKL	Delete the current spool file.	AC0: Byte pointer to device name.
.SYSTM .SPOS n	Set the current file pointer.	AC0: High-order portion of byte pointer. AC1: Low-order portion of byte pointer. AC2: Channel number, if n = 77.
.SYSTM .STAT	Get a file's statistics.	AC0: Byte pointer to filename string. AC1: Starting address of 22_8-word area.
.SYSTM .STMAP	Set the data channel map for a user device in a mapped system.	AC0: Device code. AC1: Starting user address of device buffer. Logical address of device buffer is returned.

Table A.1 RDOS command summary (continued)

Command	Description	Input and Remarks
.SYSTM .STOD	Set the time of day.	AC0: Seconds. AC1: Minutes. AC2: Hours.
.SUSP[1]	Suspend the calling task.	
.TASK	Create a task.	AC0: Task ID number in left byte. Task priority in right byte. AC1: New task entry point address. AC2: Contents passed to new task.
.TIDK	Kill a task.	AC1: Task ID number in right byte.
.TIDP	Change a task's priority.	AC0: New priority in bits 8-15. AC1: Task ID number in right byte.
.TIDR	Ready a task.	AC1: Task ID number in right byte.
.TIDS	Suspend a task.	AC1: Task ID number in right byte.
.TOVLD	Load a user overlay.	AC0: Area number in bits 0-7. Overlay number in bits 8-15. AC1: -1 for unconditional load. 0 for conditional load. AC2: Channel number on which overlay file was opened.
.TRDOP	Read a message from the console.	AC0: Byte pointer to message area (must be even). AC1: (returned) byte count.
.SYSTM .TUOFF	Turn off the tuning report function.	
.SYSTM .TUON	Turn on the tuning report function.	AC0: 0.
.TWROP	Write a message to the console.	AC0: Byte pointer to message area. AC1: -1 to suppress task ID number.
.UCEX[1,2,7]	Exit from a user clock routine.	AC1: Any nonzero value to force rescheduling.
.UIEX[1,2,7]	Exit from a user interrupt routine.	AC1: Any nonzero value to force rescheduling. AC2: In unmapped systems, value upon the call = return address. In mapped systems, unimportant.
.SYSTM .ULNK	Delete a link entry.	AC0: Byte pointer to link entry name.
.SYSTM .UPDAT n	Update the current file size.	AC2: Channel number, if n=77.

Table A.1 RDOS command summary (continued)

Command	Description	Input and Remarks
.UPIEX[1,2,7]		
.SYSTM .VMEM	Determine the number of memory blocks.	AC0: (returned) number of available blocks.
.SYSTM .WRB n	Write one or more 256-word blocks to disk.	AC0: Starting memory address. AC1: Starting relative block number. AC2: Number of disk blocks in left byte; channel number, if n=77, in right byte.[8]
.SYSTM .WRCMN	Write a message to the other program's communications area.	AC0: Word address of message. AC1: Offset into communication area. AC2: Word count.
.SYSTM .WREBL	Remove the write-protection of a memory area.	AC0: Starting address of series. AC1: Ending address of series.
.SYSTM .WRL n	Write a line.	AC0: Byte pointer to core buffer. AC1: Write byte count, including terminator, returned at end of write. AC2: Channel number, if n=77.
.SYSTM WROPR	Write an operator message.	AC0: Byte pointer to text string.
.SYSTM .WRPR	Protect an area of memory in a mapped system.	AC0: Starting address of 1K-block series. AC1: Ending address of 1K-block series.
.SYSTM .WRR n	Write a random record.	AC0: Core address of record. AC1: Record number. AC2: Channel number, if n=77.
.SYSTM .WRS n	Write sequential bytes.	AC0: Byte pointer to core buffer (must be even for MCA). AC1: Number of bytes to be written. AC2: In right byte, channel number if n=77; in left byte, number of MCA retries. (Each MCA retry takes milliseconds.)
.XMT	Transmit a message.	AC0: Message address. AC1: Message must be nonzero.
.XMTW	Transmit a message and wait.	AC0: Message address. AC1: Message must be nonzero.

Table A.1 RDOS command summary (continued)

[1] No error return.
[2] No normal return.
[3] If error EOF, error code in right byte, partial count in left byte.
[4] If error EREOF, error code in bits 8-15, partial read count in bits 0-7.
[5] Cannot be set by user.
[6] Normal return through AC2.
[7] Unmapped systems : on the interrupt, AC3 contained the return address. You must restore AC3 to this value before issuing this call.
[8] If error ERSPC, error code in right byte, partial read count in left byte.

Code	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
1	ERFNM	Illegal filename.
2	ERICM	Illegal system command.
3	ERICD	Illegal command for device.
4	ERSV1	File requires the Save attribute and the random characteristic.
6	EREOF	End of file.
7	ERRPR	Attempt to read a read-protected file.
10	ERWPR	Attempt to write a write-protected file.
11	ERCRE	Attempt to create an existing file.
12	ERDLE	Attempt to reference a nonexistent file.
13	ERDE1	Attempt to alter a permanent file.
14	ERCHA	Illegal attempt to change file attributes.
15	ERFOP	Attempt to reference an unopened file.
16	ERFUE	Fatal utility error.
17	EREXQ	Execute CLI.CM on return to CLI. (This is not really an error, but an instruction.)
20	ERNUL	Invisible error code.
21	ERUFT	Attempt to use a channel already in use.
22	ERLLI	Line limit exceeded on read or write line command.
23	ERRTN	Attempt to restore a nonexistent image.
24	ERPAR	Parity error on read line. Magnetic tape parity. (Often caused by dirty heads.)
25	ERCM3	Trying to push too many levels.
26	ERMEM	Attempt to allocate more memory than available.
27	ERSPC	Out of disk space or end of magnetic tape (EOT).

Table A.2 Error summary

Code	Mnemonic	Meaning
30	ERFIL	File read error. Magnetic tape: bad tape or odd count. Often caused by dirty heads.
31	ERSEL	Unit improperly selected.
32	ERADR	Illegal starting address.
33	ERRD	Attempt to read into system area.
34	ERDIO	Attempt to perform direct block I/O on a sequentially organized file.
35	ERDIR	Files specified on different directories.
36	ERDNM	Device not in system or illegal device code.
37	EROVN	Illegal overlay number.
40	EROVA	File not accessible by direct (free form) I/O.
41	ERTM	Attempt to set illegal time or day.
42	ERNOT	Out of TCBs.
43	ERXMT	Message address is already in use.
45	ERIBS	Interrupt device code in use.
46	ERICB	Insufficient number of free, contiguous disk blocks to create file.
47	ERSIM	Duplicate read or duplicate write to multiplexed line.
50	ERQTS	Illegal information in queue table.
51	ERNMD	Attempt to open too many devices or directories.
52	ERIDS	Illegal directory specifier.
53	ERDSN	Directory specifier unknown.
54	ER2DS	Partition is too small.
55	ERDDE	Directory depth exceeded.
56	ERDIU	Directory in use.
57	ERLDR	Link depth exceeded.
60	ERFIU	File is in use.
61	ERTID	Task ID error.
62	ERCMS	Communications area size error.

Table A.2 Error summary (continued)

Code	Mnemonic	Meaning
63	ERCUS	Communications usage error.
64	ERSCP	File position error.
65	ERDCH	Insufficient room in data channel map.
66	ERDNI	Directory or device not initialized.
67	ERNDD	No default directory.
70	ERFGE	Foreground already exists.
71	ERMPT	Error in partition set.
72	EROPD	Released directory in use by other program.
73	ERUSZ	Not enough room for UFTs within USTCH.
74	ERMPR	Address outside address space (in a mapped system only).
75	ERNLE	Attempt to delete an entry lacking the link characteristic.
76	ERNTE	Background program cannot be checkpointed.
77	ERSDE	Error detected in SYS.DR
100	ERMDE	Error detected in MAP.DR.
101	ERDTO	Device timemout.
102	ERENA	Link not allowed.
103	ERMCA	No complementary MCA request.
104	ERSRR	Short MCA receive request.
105	ERSDL	System deadlock (RDOS is out of buffers).
106	ERCLO	I/O terminated by a channel close.
107	ERSFA	Spool file is active.
110	ERABT	Task not found for abort.
111	ERDOP	Attempt to open a magnetic tape or cassette unit that is already open.
112	EROVF	System stack overflow (the current system command is aborted).
113	ERNMC	No outstanding receive request by an MCA device.

Table A.2 Error summary (continued)

Code	Mnemonic	Meaning
114	ERNIR	Attempt to initialize or release a tape unit with a currently open file.
115	ERXMZ	Attempt to transmit a zero-word message.
116	ERCANT	Gross input error, such as ECLIPSE code on a NOVA, or lowercase ASCII characters.
117	ERQOV	.TOVL not loaded for overlay task.
120	EROPM	Operator messages not specified during system generation.
121	ERFMT	Disk format error. If it recurs, dump the disk and run DKINIT.SV.
122	ERBAD	Disk has invalid bad block table. Run DKINT.SV.
123	ERBSPC	Insufficient space in core for bad block pool.
124	ERZCB	Attempt to create a contigous file of zero length.
125	ERNSE	Program is not swappable.
126	ERBLT	Blank tape.
127	ERRDY	Line not ready; modem's DSR is low (multiplexors only).
130	ERINT	Console interrupt received (multiplexors only).
131	EROVR	Hardware overrun error (multiplexors only).
132	ERFRM	Hardware framing error (multiplexors only).
133	ERSPT	Too many framing errors (DOS only, not RDOS).
134	ERPWC	Previous .WCHAR outstanding (returned from .WCHAR only).

Table A.2 Error summary (continued)

Appendix B

User Parameters

This appendix provides a listing of source file PARU.SR, which describes all RDOS user parameters. These parameters define important system calls, task calls, and mnemonics for user programs. PARU.SR was delivered with your RDOS system, along with the file PARS.SR, which contains all system parameters. Both files were loaded into your master directory during system generation.

An assembler, cross-reference listing follows the listing of parameters. Use this cross-reference to find individual parameters. The numbers in the cross-reference indicate *listing*, not appendix, pages. Parameter UFTCN, for example, has entries 1/56 and 2/05; these indicate listing page one, line 56 and listing page two, line five, respectively.

```
0001 PARU
01                          ;
02                          ; COPYRIGHT (C) DATA GENERAL CORPORATION 1977,1978,1979,1980,1982,1983
03                          ; ALL RIGHTS RESERVED.
04                          ; LICENSED MATERIAL-PROPERTY OF DATA GENERAL CORPORATION.
05                          ;
06
07                          ;= = = = = = = = = = = = = = = = = = = = = = =
08                          ; RDOS REVISION 07.10 USER PARAMETERS
09                          ;= = = = = = = = = = = = = = = = = = = = = = =
10
                                .TITL    PARU
12
13
14
15
16                          ;
17                          ; USER FILE TABLE (UFT) TEMPLATE
18                          ;
19
20                          ; USER FILE DEFINITION (UFD) OF UFT

21          000000          .DUSR UFTFN=0               ;FILE NAME
22          000005          .DUSR UFTEX=5               ;EXTENSION
23          000006          .DUSR UFTAT=6               ;FILE ATTRIBUTES
24          000007          .DUSR UFTLK=7               ;LINK ACCESS ATTRIBUTES
25          000007          .DUSR UFLAD=7               ;LINK ALTERNATE DIRECTORY
26          000010          .DUSR UFTBK=10              ;NUMBER OF LAST BLOCK IN FILE
27          000011          .DUSR UFTBC=11              ;NUMBER OF BYTES IN LAST BLOCK
28          000012          .DUSR UFTAD=12              ;DEVICE ADDRESS OF FIRST BLOCK (0 UNASSIGNED)
29          000013          .DUSR UFTAC=13              ;YEAR-DAY LAST ACCESSED
30          000014          .DUSR UFTYD=14              ;YEAR-DAY CREATED
31          000014          .DUSR UFLAN=14              ;LINK ALIAS NAME
32          000015          .DUSR UFTHM=15              ;HOUR-MINUTE CREATED
33          000016          .DUSR UFTP1=16              ;UFD TEMPORARY
34          000017          .DUSR UFTP2=17              ;WORDS/BLOCK .STAT.RSTA.CHST
35          000020          .DUSR UFTUC=20              ;USER COUNT
36          000021          .DUSR UFTDL=21              ;DCT LINK (RH) HIGH-ORDER DEVICE ADDRESS (LH)
37
38                          ; DEVICE CONTROL BLOCK (DCB) OF UFT
39
40          000022          .DUSR UFTDC=22              ;DCT ADDRESS
41          000023          .DUSR UFTUN=23              ;UNIT NUMBER
42          000024          .DUSR UFCA1=24              ;CURRENT BLOCK ADDRESS (HIGH ORDER)
43          000025          .DUSR UFTCA=25              ;CURRENT BLOCK ADDRESS (LOW ORDER)
44          000026          .DUSR UFTCB=26              ;CURRENT BLOCK NUMBER
```

```
45              000027          .DUSR UFTST=27                  ;FILE STATUS
46              000030          .DUSR UFEA1=30                  ;ENTRY'S BLOCK ADDRESS (HIGH ORDER)
47              000031          .DUSR UFTEA=31                  ;ENTRY'S BLOCK ADDRESS (LOW ORDER)
48              000032          .DUSR UFNA1=32                  ;NEXT BLOCK ADDRESS (HIGH ORDER)
49              000033          .DUSR UFTNA=33                  ;NEXT BLOCK ADDRESS (LOW ORDER)
50              000034          .DUSR UFLA1=34                  ;LAST BLOCK ADDRESS (HIGH ORDER)
51              000035          .DUSR UFTLA=35                  ;LAST BLOCK ADDRESS (LOW ORDER)
52              000036          .DUSR UFTDR=36                  ;SYS.DR DCB ADDRESS
53              000037          .DUSR UFFA1=37                  ;FIRST ADDRESS (HIGH ORDER)
54              000040          .DUSR UFTFA=40                  ;FIRST ADDRESS (LOW ORDER)
55
56                              ; DCB EXTENSION
57
58              000041          .DUSR UFTBN=41                  ;CURRENT FILE BLOCK NUMBER
59              000042          .DUSR UFTBP=42                  ;CURRENT FILE BLOCK BYTE POINTER
0002 PARU
60              000043          .DUSR UFTCH=43                  ;DEVICE CHARACTERISTICS
01              000044          .DUSR UFTCN=44                  ;ACTIVE REQ COUNT
02                                                              ;B0 INDICATES Q, 0=DSQ1,1=DSQ2
03
04
05
06
07              000045          .DUSR UFTEL=UFTCN-UFTFN+1           ;UFT ENTRY LENGTH
08              000022          .DUSR UFDEL=UFTDL-UFTFN+1           ;UFD ENTRY LENGTH
09
10              177764          .DUSR UDBAT=UFTAT-UFTDC         ;NEGATIVE DISP. TO ATTRIBUTES
11              177777          .DUSR UDDL=UFTDL-UFTDC          ;NEGATIVE DISP. TO FIRST ADDRESS (HIGH ORDER)
12              177770          .DUSR UDBAD=UFTAD-UFTDC         ;NEGATIVE DISP. TO FIRST ADDRESS (LOW ORDER)
13              177766          .DUSR UDBBK=UFTBK-UFTDC         ;NEGATIVE DISP. TO LAST BLOCK
14              000017          .DUSR UDBBN=UFTBN-UFTDC         ;POSITIVE DISP. TO CURRENT BLOCK
15
16
17
18                              ; FILE ATTRIBUTES (IN UFTAT)
19
20              100000          .DUSR ATRP  =1B0                ;READ PROTECTED
21              040000          .DUSR ATCHA=1B1                 ;CHANGE ATTRIBUTE PROTECTED
22              020000          .DUSR ATSAV=1B2                 ;SAVED FILE
23              000400          .DUSR ATNRS=1B7                 ;CANNOT BE A RESOLUTION ENTRY
24              000100          .DUSR ATUS1=1B9                 ;USER ATTRIBUTE # 1
25              000040          .DUSR ATUS2=1B10                ;USER ATTRIBUTE # 2
26              000002          .DUSR ATPER=1B14                ;PERMANENT FILE
27              000001          .DUSR ATWP  =1B15               ;WRITE PROTECTED
28
29
30                              ; FILE CHARACTERISTICS (IN UFTAT)
31
32              007400          .DUSR ATMSK=17B7                ;TO GET HIGH ORDER PART OF 3330
33                                                              ; ADDRESSES OUT OF UFTDL
34              010000          .DUSR ATLNK=1B3                 ;LINK ENTRY
35              004000          .DUSR ATPAR=1B4                 ;PARTITION ENTRY
36              002000          .DUSR ATDIR=1B5                 ;DIRECTORY ENTRY
37              001000          .DUSR ATRES=1B6                 ;LINK RESOLUTION (TEMPORARY)
38              000010          .DUSR ATCON=1B12                ;CONTIGUOUS FILE
39              000004          .DUSR ATRAN=1B13                ;RANDOM FILE
40
41                              ;
42                              ; DCT PARAMETERS.
43                              ;
44
45              000000          .DUSR DCTBS=0                   ;1B0=1 =) DEVICE USES DATA CHANNEL
46              000001          .DUSR DCTMS=1                   ;MASK OF LOWER PRIORITY DEVICES
47              000002          .DUSR DCTIS=2                   ;ADDRESS OF INTERRUPT SERVICE ROUTINE
0003 PARU
01
02                              ; DEVICE CHARACTERISTICS (IN UFTCH)
03
```

```
04          000001      .DUSR     DC100=    1B15      ; CONSOLE INPUT DEVICE IS D100 OR D200
05                                                    ; TERMINAL (SET BY INIT1)
06          000001      .DUSR     DCSTB=    1B15      ; SUPPRESS TRAILING BLANKS $CDR ONLY
07          000001      .DUSR     DCCPO=    1B15      ; DEVICE REQUIRING LEADER/TRAILER
08          000001      .DUSR     DCSTO=    1B15      ; USER SPECIFIED TIME OUT CONSTANT (MCA)
09          000002      .DUSR     DCCGN=    1B14      ; GRAPHICAL OUTPUT DEVICE WITHOUT TABBING
10                                                    ; HARDWARE
11          000004      .DUSR     DCIDI=    1B13      ; INPUT DEVICE REQUIRING OPERATOR INTERVENTION
12          000010      .DUSR     DCLCD=    1B12      ; INPUT DEVICE IS 6053-TYPE TERMINAL
13          000010      .DUSR     DCCNF=    1B12      ; OUTPUT DEVICE WITHOUT FORM FEED HARDWARE
14          000020      .DUSR     DCTO=     1B11      ; TELETYPE OUTPUT DEVICE
15          000040      .DUSR     DCKEY=    1B10      ; KEYBOARD DEVICE
16          000100      .DUSR     DCNAF=    1B09      ; OUTPUT DEVICE REQUIRING NULLS AFTER FORM FEEDS
17          000200      .DUSR     DCRAT=    1B08      ; RUBOUTS AFTER TABS REQUIRED
18          000400      .DUSR     DCPCK=    1B07      ; DEVICE REQUIRING PARITY CHECK
19          001000      .DUSR     DCLAC=    1B06      ; REQUIRES LINE FEEDS AFTER CARRIAGE RTN
20          002000      .DUSR     DCSPO=    1B05      ; SPOOLABLE DEVICE
21          004000      .DUSR     DCFWD=    1B04      ; FULL WORD DEVICE (ANYTHING GREATER THAN
22          004000      .DUSR     DCLT8=    1B04      ; LESS THAN 8 BITS / CHARACTER (BYTE DEVICES).
23          010000      .DUSR     DCFFO=    1B03      ; FORM FEEDS ON OPEN
24          020000      .DUSR     DCLTU=    1B02      ; CHANGE LOWER CASE ASCII TO UPPER
25          040000      .DUSR     DCC80=    1B01      ; READ 80 COLUMNS
26          100000      .DUSR     DCDIO=    1B00      ; SUSPEND PROTOCOL ON TRANSMIT (MCA)
27          100000      .DUSR     DCBDK=    1B00      ; DISK CHARACTERISTIC (SET NON-PARAMETRICALLY)
28                                                    ; SET MEANS ITS 3330
29          100000      .DUSR     DCSPC=    1B00      ; SPOOL CONTROL
30                                                    ; SET = SPOOLING ENABLED
31                                                    ; RESET = SPOOLING DISABLED
32
33                      ;CHARACTERISTICS WORD FOR MY DCT'S
34
35          143432      .DUSR     TTOCH=DCCGN+DCCNF+DCTO+DCPCK+DCLAC+DCC80+DCSPO+DCSPC
36          020040      .DUSR     TTICH=DCKEY+DCLTU
37
0004 PARU
01                      ;
02                      ; DEVICE CHARACTERISTICS FOR QTY, ULM, AND ALM (PARU.SR)
03                      ;
04          000001       .DUSR    DCNI=     1B15      ;(MASKING ENABLES) CONSOLE INTERRUPTS
05                      ;.DUSR    DCCGN=    1B14      ;(MASKING DISABLES) TAB EXPANSION
06          000004       .DUSR    DCLOC=    1B13      ;LOCAL LINE (MASKING MAKES MODEM LINE)
07
08                      ;.DUSR    DCTO=     1B11      ;_ FOR RUBOUT (MASKING GIVES BACKSPACE)
09                                                    ;IGNORE LINEFEED (MASKING CONVERTS
10                                                    ; LF/NL TO CR)
11                      ;.DUSR    DCKEY=    1B10      ;(MASKING DISABLES) INPUT ECHOING.
12                                                    ;MASKING ALSO DISABLES LINE EDIT
13                                                    ;( Z,ESC,DEL,\ ),
14                                                    ;UNLESS "DCEDT" ALSO MASKED.
15
16                      ;.DUSR    DCNAF=    1B9       ;(MASKING DISABLES) 20 NULLS AFTER FORM FEED
17          000200       .DUSR    DCXON=    1B8       ;(MASKING ENABLES) XON/XOFF FOR $TTR
18                      ;                   1B7       ;SAVE FOR FUTURE USE
19
20                      ;.DUSR    DCLAC=    1B6       ;(MASKING DISABLES) LINE FEED AFTER
21                                                    ;  CARRIAGE RETURN
22                      ;.DUSR    DCSPO=    1B5       ;(MUST BE OFF) SPOOLING
23          004000       .DUSR    DCCRE=    1B4       ;CARRIAGE RETURN ECHO (MASKING DISABLES)
24          100000       .DUSR    DCEDT=    1B0       ;LINE EDIT (ESC, Z,DEL, \) DISABLED IF
25                                                    ; MASK THIS BIT OR "DCKEY", BUT NOT BOTH.
26                      ;
27                      ; .WRL TO QTY:64
28                      ;
29                      ;          AC0= CODE+LINE #
30                      ;          AC1= DATA
31
32          000000      .DUSR     W64DC=    0B7       ;NEW DEVICE CHARACTERISTIC MASK
33                                                          ;FOR OPEN CHANNEL, AC1 AS ABOVE.
```

```
34              000400          .DUSR       W64LS=    1B7       ;CHANGE LINE SPEED FOR DG/CS,
35                                                                ; AC1 RIGHT-JUSTIFIED CLOCK SELECT.
36              001000          .DUSR       W64MS=    2B7       ;CHANGE DG/CS MODEM STATE, AC1 =
37              000001          .DUSR         W64DTR= 1B15              ;  RAISE DATA TERMINAL READY
38                                                                ;     ELSE LOWER
39              000002          .DUSR         W64RTS= 1B14              ;  RAISE REQUEST TO SEND
40                                                                ;     ELSE LOWER
41              001400          .DUSR       W64CH=    3B7       ;CHANGE CHARACTERISTICS FOR LINE
42                                                                ;AC1 SAME AS DG/CS HARDWARE SPEC.
43
0005 PARU
01                              ;
02                              ; SWITCHES
03                              ;
04
05              100000          .DUSR       A.SW=     1B00
06              040000          .DUSR       B.SW=     1B01
07              020000          .DUSR       C.SW=     1B02
08              010000          .DUSR       D.SW=     1B03
09              004000          .DUSR       E.SW=     1B04
10              002000          .DUSR       F.SW=     1B05
11              001000          .DUSR       G.SW=     1B06
12              000400          .DUSR       H.SW=     1B07
13              000200          .DUSR       I.SW=     1B08
14              000100          .DUSR       J.SW=     1B09
15              000040          .DUSR       K.SW=     1B10
16              000020          .DUSR       L.SW=     1B11
17              000010          .DUSR       M.SW=     1B12
18              000004          .DUSR       N.SW=     1B13
19              000002          .DUSR       O.SW=     1B14
20              000001          .DUSR       P.SW=     1B15
21              100000          .DUSR       Q.SW=     1B00
22              040000          .DUSR       R.SW=     1B01
23              020000          .DUSR       S.SW=     1B02
24              010000          .DUSR       T.SW=     1B03
25              004000          .DUSR       U.SW=     1B04
26              002000          .DUSR       V.SW=     1B05
27              001000          .DUSR       W.SW=     1B06
28              000400          .DUSR       X.SW=     1B07
29              000200          .DUSR       Y.SW=     1B08
30              000100          .DUSR       Z.SW=     1B09
0006 PARU
01
02                              ;
03                              ; SYSTEM CONSTANTS
04                              ;
05
06              000377          .DUSR SCWPB=255.                  ;WORDS PER BLOCK
07              000400          .DUSR SCDBS=256.                  ;SIZE OF DISK BLOCK
08              000100          .DUSR SCRRL=64.                   ;WORDS PER RANDOM RECORD
09              000204          .DUSR SCLLG=132.                  ;MAX LINE LENGTH
10              000030          .DUSR SCAMX=24.                   ;MAX ARGUMENT LENGTH IN BYTES
11              000006          .DUSR SCFNL=UFTEX-UFTFN+1         ;FILE NAME LENGTH
12              000005          .DUSR SCEXT=UFTEX-UFTFN           ;EXTENSION OFFSET IN NAME AREA
13              000012          .DUSR SCMER=10.                   ;MAX ERROR RETRY COUNT
14              000016          .DUSR SCSTR=16                    ;SAVE FILE STARTING ADDRESS
15              177660          .DUSR SCTIM=-80.                  ;RINGIO 1 MS. LOOP TIME (SN)
16              000000          .DUSR SCPPL=0                     ;PRIMARY PARTITION LEVEL
17              000006          .DUSR SCPPA=6                     ;PRIMARY PARTITION BASE ADDRESS
18              000003          .DUSR SCDSK=3                     ;ABSOLUTE ADDRESS OF DISK INFORMATION BLOCK
19              000004          .DUSR SCBAD=4                     ;ABSOLUTE ADDRESS OF BAD BLOCK TABLE BLOCK
20              000000          .DUSR SCSYS=0                     ;SYS.DR ADDRESS OFFSET
21              000001          .DUSR SCPSH=1                     ;PUSH DIRECTORY OFFSET
22              000004          .DUSR SCPNM=4                     ;MAX NUMBER OF PUSH LEVELS
23              000011          .DUSR SCMAP=SCPNM*2+SCPSH         ;RELATIVE BASE ADDRESS OF MAP.DR
24              000001          .DUSR SCBPB=1                     ;RELATIVE BACKGROUND PUSH BASE
25              000006          .DUSR SCFPB=SCBPB+SCPNM+1         ;RELATIVE FOREGROUND PUSH BASE
26              000021          .DUSR SCFZW=SCPNM*4+SCBPB         ;FRAME SIZE WORD (SKIP DOUBLE WORD PUSH INDICES)
```

```
27              000022          .DUSR SCNVW=SCFZW+1             ;NUMBER-OF-SYSTEM-OVERLAYS WORD
28              100000          .DUSR SFINT=1B0                 ;INTERRUPT FLAG
29              000001          .DUSR SFBRK=1B15                ;BREAK FLAG
30              000100          .DUSR SCNSO=64.                 ;NUMBER OF SYSTEM OVERLAYS
31              000072          .DUSR SNSOU=72                  ;NUMBER OF SYSTEM OVERLAYS IN USE
32              000017          .DUSR SCSOP=SNSOU+3/4           ;NUMBER OF MEMORY PAGES TO HOLD SYSTEM OVERLAYS
33
34
35                              ; SYSTEM BOOTSTRAP CONSTANTS
36
37              000000          .DUSR  SCTBP=0                  ;TEXT STRING BYTE POINTER
38              000001          .DUSR  SCINS=1                  ;SWITCHED FULL/PARTIAL-OVERLAYS ADDRESS
39              000002          .DUSR  SCPSA=2                  ;PROGRAM START ADDRESS
40              000002          .DUSR  SCPAR=SCPSA              ;PARTIAL INIT ADDRESS
41              000003          .DUSR  SCINT=3                  ;FULL/PARTIAL-OVERLAYS INIT ADDRESS
42              000004          .DUSR  SCCLI=SCINT+1            ;ADDRESS OF END OF CLI
43              000005          .DUSR  SCZMX=SCCLI+1            ;SQUASHED/UNSQUASHED FLAG
44              000006          .DUSR  SCCPL=SCZMX+1            ;CURRENT PARTITION LEVEL
45              000007          .DUSR  SCPBA=SCCPL+1            ;PARTITION BASE ADDRESS (LOW ORDER)
46              000010          .DUSR  SCOFA=SCPBA+1            ;OVERLAY BASE ADDRESS (LOW ORDER)
47              000011          .DUSR  SCPB1=SCOFA+1            ;PARTITION BASE ADDRESS (HIGH ORDER)
48              000012          .DUSR  SCOF1=SCPB1+1            ;OVERLAY BASE ADDRESS (HIGH ORDER)
49              000013          .DUSR  SCBAS=SCOF1+1            ;BASE OF INFORMATION BLOCK
50              000013          .DUSR  SCSWC=SCBAS              ;SWITCH FOR SCINS ENTRY
51              000020          .DUSR  SCIDV=20                 ;INITIAL DEVICE CODE
52
53              000000          .DUSR  SCAUN=0                  ;ASCII UNIT NUMBER
54              000001          .DUSR  SCUN=1                   ;UNIT (DEVICE CODE)
55              000002          .DUSR  SCGO=2                   ,ENTRY TO PASS FILENAME
56              000004          .DUSR  SCNGO=4                  ;ENTRY TO ASK FROM CONSOLE
0007 PARU
01
02                              ; SYSTEM ERROR CODES
03
04              000000          .DUSR ERFNO=    0       ; ILLEGAL CHANNEL NUMBER
05              000001          .DUSR ERFNM=    1       ; ILLEGAL FILE NAME
06              000002          .DUSR ERICM=    2       ; ILLEGAL SYSTEM COMMAND
07              000003          .DUSR ERICD=    3       ; ILLEGAL COMMAND FOR DEVICE
08              000004          .DUSR ERSV1=    4       ; NOT A SAVED FILE
09              000005          .DUSR ERWR0=    5       ; ATTEMPT TO WRITE AN EXISTENT FILE
10              000006          .DUSR EREOF=    6       ; END OF FILE
11              000007          .DUSR ERRPR=    7       ; ATTEMPT TO READ A READ PROTECTED FILE
12              000010          .DUSR ERWPR=    10      ; WRITE PROTECTED FILE
13              000011          .DUSR ERCRE=    11      ; ATTEMPT TO CREATE AN EXISTENT FILE
14              000012          .DUSR ERDLE=    12      ; A NON-EXISTENT FILE
15              000013          .DUSR ERDE1=    13      ; ATTEMPT TO ALTER A PERMANENT FILE
16              000014          .DUSR ERCHA=    14      ; ATTRIBUTES PROTECTED
17              000015          .DUSR ERFOP=    15      ; FILE NOT OPENED
18              000016          .DUSR ERFUE=    16      ; FATAL UTILITY ERROR
19              000017          .DUSR  EREXQ=   17      ; EXECUTE CLI.CM (NO ERROR)
20              000020          .DUSR  ERNUL=   20      ; INVISIBLE ERROR CODE
21              000021          .DUSR ERUFT=    21      ; ATTEMPT TO USE A UFT ALREADY IN USE
22              000022          .DUSR ERLLI=    22      ; LINE LIMIT EXCEEDED O
23              000023          .DUSR ERRTN=    23      ; ATTEMPT TO RESTORE A NON-EXISTENT IMAGE
24              000024          .DUSR ERPAR=    24      ; PARITY ERROR ON READ LINE
25              000025          .DUSR ERCM3=    25      ; TRYING TO PUSH TOO MANY LEVELS
26              000026          .DUSR ERMEM=    26      ; NOT ENUF MEMORY AVAILABLE
27              000027          .DUSR ERSPC=    27      ; OUT OF FILE SPACE
28              000030          .DUSR ERFIL=    30      ; FILE READ ERROR
29              000031          .DUSR ERSEL=    31      ; UNIT NOT PROPERLY SELECTED
30              000032          .DUSR ERADR=    32      ; ILLEGAL STARTING ADDRESS
31              000033          .DUSR ERRD=     33      ; ATTEMPT TO READ INTO SYSTEM AREA
32              000034          .DUSR ERDIO=    34      ; FILE ACCESSIBLE BY DIRECT I/O ONLY
33              000035          .DUSR ERDIR=    35      ; FILES SPECIFIED ON DIFF. DIRECTORIES
34              000036          .DUSR ERDNM=    36      ; DEVICE NOT IN SYSTEM
35              000037          .DUSR EROVN=    37      ; ILLEGAL OVERLAY NUMBER
36              000040          .DUSR EROVA=    40      ; FILE NOT ACCESSIBLE BY DIRECT I/O
37              000041          .DUSR ERTIM=    41      ; USER SET TIME ERROR
```

```
38          000042      .DUSR ERNOT=      42      ; OUT OF TCB'S
39          000043      .DUSR ERXMT=      43      ; SIGNAL TO BUSY ADDR
40          000044      .DUSR ERSQF=      44      ; FILE ALREADY SQUASHED ERROR
41          000045      .DUSR ERIBS=      45      ; DEVICE ALREADY IN SYSTEM
42          000046      .DUSR ERICB=      46      ; INSUFFICENT CONTIGUOUS BLOCKS
43          000047      .DUSR ERSIM=      47      ; SIMULTANEOUS READ OR WRITE TO MUX LINE
44          000050      .DUSR ERQTS=      50      ; ERROR IN USER TASK QUEUE TABLE
45          000051      .DUSR ERNMD=      51      ; NO MORE DCB'S
46          000052      .DUSR ERIDS=      52      ; ILLEGAL DIRECTORY SPECIFIER
47          000053      .DUSR ERDSN=      53      ; DIRECTORY SPECIFIER NOT KNOWN
48          000054      .DUSR ERD2S=      54      ; DIRECTORY IS TOO SMALL
49          000055      .DUSR ERDDE=      55      ; DIRECTORY DEPTH EXCEEDED
50          000056      .DUSR ERDIU=      56      ; DIRECTORY IN USE
51          000057      .DUSR ERLDE=      57      ; LINK DEPTH EXCEEDED
52          000060      .DUSR ERFIU=      60      ; FILE IS IN USE
53          000061      .DUSR ERTID=      61      ; TASK ID ERROR
54          000062      .DUSR ERCMS=      62      ; COMMON SIZE ERROR
55          000063      .DUSR ERCUS=      63      ; COMMON USAGE ERROR
56          000064      .DUSR ERSCP=      64      ; FILE POSITION ERROR
57          000065      .DUSR ERDCH=      65      ; INSUFFICIENT ROOM IN DATA CHANNEL MAP
58          000066      .DUSR ERDNI=      66      ; DIRECTORY NOT INITIALIZED
59          000067      .DUSR ERNDD=      67      ; NO DEFAULT DIRECTORY
60          000070      .DUSR ERFGE=      70      ; FOREGROUND ALREADY EXISTS
0008 PARU
01          000071      .DUSR ERMPT=      71      ; ERROR IN PARTITON SET
02          000072      .DUSR EROPD=      72      ; DIRECTORY IN USE BY OTHER PROGRAM
03          000073      .DUSR ERUSZ=      73      ; NO ROOM FOR UFTS ON EXEC/EXFG
04          000074      .DUSR ERMPR=      74      ; ADDR ERROR ON .SYSTM PARAM
05          000075      .DUSR ERNLE=      75      ; NOT A LINK ENTRY
06          000076      .DUSR ERNTE=      76      ; CURRENT BG IS NOT CHECKPOINTABLE
07          000077      .DUSR ERSDE=      77      ; SYS.DR ERROR
08          000100      .DUSR ERMDE=      100     ; MAP.DR ERROR
09          000101      .DUSR ERDTO=      101     ; DEVICE TIME OUT
10          000102      .DUSR ERENA=      102     ; ENTRY NOT ACCESSIBLE VIA LINK
11          000103      .DUSR ERMCA=      103     ; MCA REQUEST OUTSTANDING
12          000104      .DUSR ERSRR=      104     ; INCOMPLETE TRANSMISSION CAUSED BY RECIEVER
13          000105      .DUSR ERSDL=      105     ; SYSTEM DEADLOCK
14          000106      .DUSR ERCLO=      106     ; I/O TERMINATED BY CHANNEL CLOSE
15          000107      .DUSR ERSFA=      107     ; SPOOL FILE(S) ACTIVE
16          000110      .DUSR ERABT=      110     ; TASK NOT FOUND FOR ABORT
17          000111      .DUSR ERDOP=      111     ; DEVICE PREVIOUSLY OPENED
18          000112      .DUSR EROVF=      112     ; SYSTEM STACK OVERFLOW
19          000113      .DUSR ERNMC=      113     ; NO MCA RECEIVE REQUEST OUTSTANDING
20          000114      .DUSR ERNIR=      114     ; NO INIT/RELEASE ON OPENED DEVICE (MAG TAPE)
21          000115      .DUSR ERXMZ=      115     ; .XMT & .IXMT MESSAGES MUST BE NON-ZERO
22          000116      .DUSR ERCANT=     116     ; 'YOU CAN'T DO THAT'
23          000117      .DUSR ERQOV=      117     ; .TOVLD NOT LOADED FOR QUEUED OVERLAY TASKS
24          000120      .DUSR EROPM=      120     ; OPERATOR MESSAGE MODULE NOT SYSGENED
25          000121      .DUSR ERFMT=      121     ; DISK FORMAT ERROR
26          000122      .DUSR ERBAD=      122     ; DISK HAS INVALID BAD BLOCK TABLE
27          000123      .DUSR ERBSPC=     123     ; INSUFFICIENT SPACE IN BAD BLOCK POOL (CORE)
28          000124      .DUSR ERZCB=      124     ; ATTEMPT TO CREATE CONTIG OF ZERO LENGTH
29          000125      .DUSR ERNSE=      125     ; PROGRAM IS NOT SWAPPABLE
30          000126      .DUSR ERBLT=      126     ; BLANK TAPE
31          000127      .DUSR ERRDY=      127     ; LINE NOT READY
32          000130      .DUSR ERINT=      130     ; CONSOLE INTERRUPT RECEIVED
33          000131      .DUSR EROVR=      131     ; CHARACTER OVER RUN ERROR
34          000132      .DUSR ERFRM=      132     ; CHARACTER FRAMING ERROR
35          000133      .DUSR ERSPT=      133     ; TOO MANY SOFT ERRORS (DOS ONLY)
36          000134      .DUSR ERSOF=      134     ; QTY BUFFER OVERFLOW
0009 PARU
01
02                      ; CLI ERROR CODES
03
04          000300      .DUSR   CNEAR=    300     ; NOT ENOUGH ARGUMENTS
05          000301      .DUSR   CILAT=    301     ; ILLEGAL ATTRIBUTE
06          000302      .DUSR   CNDBD=    302     ; NO DEBUG ADDRESS
07          000303      .DUSR   CCLTL=    303     ; COMMAND LINE TOO LONG
```

```
08          000304      .DUSR     CNSAD=    304       ; NO STARTING ADDRESS
09          000305      .DUSR     CCKER=    305       ; CHECKSUM ERROR
10          000306      .DUSR     CNSFS=    306       ; NO SOURCE FILE SPECIFIED
11          000307      .DUSR     CNACM=    307       ; NOT A COMMAND
12          000301      .DUSR     CILBK=    310       ; ILLEGAL BLOCK TYPE
13          000311      .DUSR     CSPER=    311       ; NO FILES MATCH SPECIFIER
14          000312      .DUSR     CPHER=    312       ; PHASE ERROR
15          000313      .DUSR     CTMAR=    313       ; TOO MANY ARGUMENTS
16          000314      .DUSR     CTMAD=    314       ; TOO MANY ACTIVE DEVICES
17          000315      .DUSR     CILNA=    315       ; ILLEGAL NUMERIC ARGUMENT
18          000316      .DUSR     CSFUE=    316       ; FATAL SYSTEM UTILITY ERROR
19          000317      .DUSR     CILAR=    317       ; ILLEGAL ARGUMENT
20          000320      .DUSR     CCANT=    320       ; IMPROPER OR MALICIOUS INPUT
21          000321      .DUSR     CTMLI=    321       ; TOO MANY LEVELS OF INDIRECT FILES
22          000322      .DUSR     CSYER=    322       ; SYNTAX ERROR
23          000323      .DUSR     CBKER=    323       ; BRACKET ERROR
24          000324      .DUSR     CPARE=    324       ; PAREN ERROR
25          000325      .DUSR     CCART=    325       ; < WITHOUT > OR > WITHOUT <
26          000326      .DUSR     CCAR1=    326       ; ILLEGAL NESTING OF <> AND ()
27          000327      .DUSR     CINDE=    327       ; ILLEGAL INDIRECT FILENAME
28          000330      .DUSR     CPAR1=    330       ; ILLEGAL NESTING OF () AND []
29          000331      .DUSR     CIVAR=    331       ; ILLEGAL VARIABLE
30          000332      .DUSR     CILTA=    332       ; ILLEGAL TEXT ARGUMENT
31          000333      .DUSR     CTATL=    333       ; TEXT ARGUMENT TOO LONG
32
33          000333      .DUSR     CCMAX=    CTATL     ; MAX CLI ERROR CODE
34          000036      .DUSR     ERML=     30.       ; MAXIMUM ERROR MESSAGE LENGTH
35
36
37
38
39                      ; EXCEPTIONAL SYSTEM STATUS CODES
40
41          100001      .DUSR     PNMPF=    @1        ; MAP DR ERROR
42          100002      .DUSR     PNSDE=    @2        ; SYSTEM DIRECTORY ERROR
43          100003      .DUSR     PNCSO=    @3        ; SYSTEM STACK FAULT
44          100004      .DUSR     PNIDA=    @4        ; INCONSISTENT SYSTEM DATA
45          100005      .DUSR     PNMDD=    @5        ; MASTER DEVICE DATA ERROR
46          100006      .DUSR     PNMDT=    @6        ; MASTER DEVICE TIME OUT
47          100007      .DUSR     PNDPE=    @7        ; MOVING HEAD DISK ERROR
48          100010      .DUSR     PNCUI=    @10       ; UNCLEARABLE UNDEFINED INTERRUPT
49          100012      .DUSR     PNCBK=    @12       ; INSUFFICENT CONTIGUOUS BLOCKS TO BUILD
50                                                    ; PUSH SPACE INDICES
51          100011      .DUSR     PNILL=    @11       ; ILLEGAL EXTENDED INSTRUCTION
52          100013      .DUSR     PNPSH=    @13       ; RTN BEYOND TOP OF WORLD
53          100014      .DUSR     PNIPB=    @14       ; INCONSISTENT OR IMPOSSIBLE CONDITION
54                                                    ; RELATED TO DUAL PROCESSORS (IPB)
55          100015      .DUSR     PNITR=    @15       ; INT WORLD TRAPPED
56          100016      .DUSR     PNERC=    @16       ; MULTIBIT MEMORY ERROR
57          100017      .DUSR     PNPAR=    @17       ; MEMORY PARITY ERROR
58          100020      .DUSR     PNMEM=    @20       ; INFOS INSUFFICIENT MEMORY (INIT TIME)
59          100021      .DUSR     PNSPL=    @21       ; SPOOLER
60          100022      .DUSR     PNEMT=    @22       ;MICRO-ECLIPSE EMULATOR TRAP
0010 PARU
01                                                    ;PANIC -- UNMAPPED RDOS ONLY !!!!
02          100023      .DUSR     PNPSF=    @23       ;POWER SUPPLY FAULT-UPSC
0011 PARU
01
02                      ;PANIC CODES MADE FROM GENERIC EXCEPTIONAL SYSTEM STATUS CODES
03                      ;#R013 ED6 8/31/79
04                      ;
05
06                      ;         PANIC          STAT          MODULE
07
08          110002      .DUSR     P1SDE     = 1B3!PNSDE       ;WDBLK
09
10          110003      .DUSR     P1CSO     = 1B3!PNCSO       ;GSUB
11          120003      .DUSR     P2CSO     = 2B3!PNCSO       ;INTD
```

```
12          130003      .DUSR   P3CSO   = 3B3!PNCSO         ;GSUB
13          140003      .DUSR   P4CSO   = 4B3!PNCSO         ;GSUB
14          150003      .DUSR   P5CSO   = 5B3!PNCSO         ;GSUB
15
16          110004      .DUSR   P1IDA   = 1B3!PNIDA         ;FILIO
17          120004      .DUSR   P2IDA   = 2B3!PNIDA         ;BLKIO
18          130004      .DUSR   P3IDA   = 3B3!PNIDA         ;IOBUF
19          140004      .DUSR   P4IDA   = 4B3!PNIDA         ;OPPRO
20
21          110005      .DUSR   P1MDD   = 1B3!PNMDD         ;OVLAY
22          120005      .DUSR   P2MDD   = 2B3!PNMDD         ;INIT3
23          130005      .DUSR   P3MDD   = 3B3!PNMDD         ;SOV3
24          140005      .DUSR   P4MDD   = 4B3!PNMDD         ;SOV6
25          150005      .DUSR   P5MDD   = 5B3!PNMDD         ;SOV7
26          160005      .DUSR   P6MDD   = 6B3!PNMDD         ;SOV3
27
28
29          170005      .DUSR   P7MDD   = 7B3!PNMDD         ;FILIO
30          110006      .DUSR   P1MDT   = 1B3!PNMDT         ;INIT3
31          120006      .DUSR   P2MDT   = 2B3!PNMDT         ;OVLAY
32          130006      .DUSR   P3MDT   = 3B3!PNMDT         ;SOV3
33          140006      .DUSR   P4MDT   = 4B3!PNMDT         ;SOV6
34          150006      .DUSR   P5MDT   = 5B3!PNMDT         ;SOV7
35          160006      .DUSR   P6MDT   = 6B3!PNMDT         ;SOV3
36
37          110007      .DUSR   P1DPE   = 1B3!PNDPE         ;DZPDR
38          110010      .DUSR   P1CUI   = 1B3!PNCUI         ;INTD
39
40          110012      .DUSR   P1CBK   = 1B3!PNCBK         ;FINIT2
41          120012      .DUSR   P2CBK   = 2B3!PNCBK         ;INIT2
42
43          110011      .DUSR   P1ILL   = 1B3!PNILL         ;GSUB
44
45          110013      .DUSR   P1PSH   = 1B3!PNPSH         ;SOV12
46          120013      .DUSR   P2PSH   = 2B3!PNPSH         ;SOV27
47
48          110014      .DUSR   P1IPB   = 1B3!PNIPB         ;DPMOD
49          120014      .DUSR   P2IPB   = 2B3!PNIPB         ;DPMOD
50          130014      .DUSR   P3IPB   = 3B3!PNIPB         ;DPMOD
51          140014      .DUSR   P4IPB   = 4B3!PNIPB         ;DPMOD
52
53          110015      .DUSR   P1ITR   = 1B3!PNITR         ;MAPZ
54          120015      .DUSR   P2ITR   = 2B3!PNITR         ;MAPZ
55
56          110016      .DUSR   P1ERC   = 1B3!PNERC         ;INTD
57
58          110017      .DUSR   P1PAR   = 1B3!PNPAR         ;INTD
59
60          110021      .DUSR   P1SPL   = 1B3!PNSPL         ;SPOLR
0012 PARU
01          110022      .DUSR   P1EMT   = 1B3!PNEMT         ;PANIC
02          110023      .DUSR   P1PSF   = 1B3!PNPSF         ;UPSC
03
0013 PARU
01
02                      ;
03                      ; USER STATUS TABLE (UST) TEMPLATE
04                      ;
05          000400      .DUSR   UST =     400       ; START OF BACKGROUND USER STATUS AREA
06
07          000012      .DUSR   USTP=12             ; PZERO LOC FOR UST POINTER
08                      ; NOTE-   USTP MUST CORRESPOND TO PARS PZERO ALLOCATIONS
09
10          000000      .DUSR   USTPC=    0         ; 0=)BACKGROUND, 1=)FOREGROUND
11                                                  ; (WHEN NOT IN SCHED STATE)
12          000001      .DUSR   USTZM=    1         ; ZMAX
13          000002      .DUSR   USTSS=    2         ; START OF SYMBOL TABLE
14          000003      .DUSR   USTES=    3         ; END OF SYMBOL TABLE
```

```
15          000004      .DUSR    USTNM=   4          ; NMAX
16          000005      .DUSR    USTSA=   5          ; STARTING ADDRESS
17          000006      .DUSR    USTDA=   6          ; DEBUGGER ADDRESS
18          000007      .DUSR    USTHU=   7          ; HIGHEST ADDRESS USED
19          000010      .DUSR    USTCS=   10         ; FORTRAN COMMON AREA SIZE
20          000011      .DUSR    USTIT=   11         ; INTERRUPT ADDRESS
21          000012      .DUSR    USTBR=   12         ; BREAK ADDRESS
22          000013      .DUSR    USTCH=   13         ; # TASKS (LEFT), # CHANS (RIGHT)
23          000014      .DUSR    USTCT=   14         ; CURRENTLY ACTIVE TCB
24          000015      .DUSR    USTAC=   15         ; START OF ACTIVE TCB CHAIN
25          000016      .DUSR    USTFC=   16         ; START OF FREE TCB CHAIN
26          000017      .DUSR    USTIN=   17         ; INITIAL START OF NREL
27          000020      .DUSR    USTOD=   20         ; OVLY DIRECTORY ADDR
28          000021      .DUSR    USTSV=   21         ; FORTRAN STATE VARIABLE SAVE ROUTINE (OR 0)
29          000022      .DUSR    USTRV=   22         ; REVISION
30                                                   ; ENVIRONMENT STATE WORD WHEN EXECUTING
31          000023      .DUSR    USTIA=   23         ; TCB ADDR OF INT OR BREAK PROC
32
33          000023      .DUSR    USTEN=   USTIA      ; LAST ENTRY
34
35          000030      .DUSR    UFPT=    30         ; SAVE SOS
36
37
38
39                      ; ENVIRONMENT STATUS BITS (IN USTRV DURING EXECUTION)
40
41          100000      .DUSR    ENMAP=   1B0        ;MAPPED MACHINE
42          020000      .DUSR    ENUEC=   1B2        ;UNMAPPED ECLIPSE
43          010000      .DUSR    ENMEC=   1B3        ;MAPPED ECLIPSE
44          004000      .DUSR    ENUNV=   1B4        ,UNMAPPED NOVA
45          002000      .DUSR    ENMNV=   1B5        ;MAPPED NOVA
46          001000      .DUSR    ENUN3=   1B6        ;UNMAPPED NOVA 3
47          000400      .DUSR    ENMN3=   1B7        ;MAPPED NOVA 3
48          000200      .DUSR    ENUMN=   1B8        ;UNMAPPED MICRO NOVA
49
50          000100      .DUSR    ENCTD=   1B9        ;#MSB#  D FROM TTY FOR ICOS SYSTEMS
51          000020      .DUSR    ENDOS=   1B11       ;DOS SYSTEM
52          000010      .DUSR    ENINFO=  1B12       ;INFOS SYSTEM
53          000004      .DUSR    ENSOS=   1B13       ;STAND ALONE SYSTEM
54          000002      .DUSR    ENRTOS=  1B14       ;RTOS SYSTEM
55          000001      .DUSR    ENRDOS=  1B15       ;RDOS SYSTEM
0014 PARU
01
02                      ;
03                      ; TASK CONTROL BLOCK (TCB) TEMPLATE
04                      ;
05
06          000000      .DUSR TPC=          0        ;USER PC (B0-14) + CARRY (B15)
07          000001      .DUSR TAC0=         1        ;AC0
08          000002      .DUSR TAC1=         2        ;AC1
09          000003      .DUSR TAC2=         3        ;AC2
10          000004      .DUSR TAC3=         4        ;AC3
11          000005      .DUSR TPRST=        5        ;STATUS BITS (LEFT) + PRIORITY (RIGHT)
12          000006      .DUSR TSYS=         6        ;SYSTEM CALL WORD
13          000007      .DUSR TLNK=         7        ;LINK WORD
14          000010      .DUSR TUSP=         10       ;USP
15          000011      .DUSR TELN=         11       ;TCB EXTENSION ADDR
16          000012      .DUSR TID=          12       ;TASK ID
17          000013      .DUSR TTMP=         13       ;SCHEDULER TEMPORARY
18          000014      .DUSR TKLAD=        14       ;USER KILL PROC ADDR
19          000015      .DUSR  TSP=         15       ;STACK POINTER
20          000016      .DUSR  TFP=         16       ;FRAME POINTER
21          000017      .DUSR  TSL=         17       ;STACK LIMIT
22          000020      .DUSR  TSO=         20       ;OVERFLOW ADDR
23
24          000015      .DUSR TLN=TKLAD-TPC+1        ;SHORT TCB LENGTH
25          000021      .DUSR TLNB=TSO-TPC+1         ;LONG TCB LENGTH
26
27
```

```
28                      ; TASK STATUS BITS (IN TPRST)
29
30          100000      .DUSR   TSSYS=      1B0         ;SYSTEM BIT
31          040000      .DUSR   TSSUSP=     1B1         ;SUSPEND BIT
32          020000      .DUSR   TSXMT=      1B2         ;XMT/REC AND OVERLAY BIT
33          010000      .DUSR   TSRDOP=     1B3         ;.TRDOP BIT
34          004000      .DUSR   TSABT=      1B4         ;ABORT LOCK BIT
35          002000      .DUSR   TSRSV=      1B5         ;RESERVED
36          001000      .DUSR   TSUPN=      1B6         ;USER PEND BIT
37          000400      .DUSR   TSUSR=      1B7         ;USER FLAG BIT
0015 PARU
01                      ;
02                      ; OVERLAY DIRECTORY
03                      ;
04
05          000000      .DUSR       OVNDS=  0           ;NUMBER OF NODES
06
07                      ; FOR EACH NODE:
08
09          000001      .DUSR       OVRES=  1           ;CURRENT OVLY(B0-7), USE COUNT(B8-15)
10          000002      .DUSR       OVDIS=  2           ;# OVLYS (B0-7), LOADING BIT (B8),
11                                                      ; SIZE IN BLKS (B9-15)
12          000003      .DUSR       OVBLK=  3           ;STRT BLK # IN OVLY FILE FOR FIRST OVLY
13          000004      .DUSR       OVNAD=  4           ;CORE ADDR FOR NODE(B1-15)
14                                                      ; 1B0 FLAGS VIRTUAL NODE
15
16
17
18                      ;
19                      ; USER TASK QUEUE TABLE
20                      ;
21
22          000000      .DUSR       QPC=    0           ;STARTING PC
23          000001      .DUSR       QNUM=   1           ;NUMBER OF TIMES TO EXEC
24          000002      .DUSR       QTOV=   2           ;OVERLAY
25          000003      .DUSR       QSH=    3           ;STARTING HOUR
26          000004      .DUSR       QSMS=   4           ;STARTING SEC IN HOUR
27          000005      .DUSR       QPRI=   TPRST       ;MUST BE SAME
28          000006      .DUSR       QRR=    6           ;RERUN TIME INC IN SEC
29          000007      .DUSR       QTLNK=  TLNK        ;MUST BE SAME
30          000010      .DUSR       QOCH=   10          ;CHAN OVERLAYS OPEN ON
31          000011      .DUSR       QCOND=  11          ;TYPE OF LOAD
32          000012      .DUSR       QAC2=   12          ;WAKEUP AC2
33                                                      ; 1B0= LOADING, 1B15= DEQUE REQ REC
34          000013      .DUSR       QTLN=   QAC2-QPC+1
35          000013      .DUSR       QPEX=   QTLN        ;USER TASK Q AREA EXTENSION
36
37
38
39                      ;
40                      ; USER PROGRAM TABLE FOR OPERATOR COMMUNICATIONS PACKAGE
41                      ;
42
43          000000      .DUSR       LPN=    0           ;PROGRAM NUMBER
44          000001      .DUSR       LOV=    1           ;OVERLAY NUMBER OR -1
45          000002      .DUSR       LCOND=  2           ;CONDITIONAL/UNCONDITIONAL LOAD
46          000003      .DUSR       LTPR=   3           ;TASK ID (LEFT) + PRIORITY (RIGHT)
47          000004      .DUSR       LPC=    4           ;PROGRAM COUNTER
48
49          000005      .DUSR       LTLN= LPC-LPN+1     ;TABLE LENGTH
50
51          000005      .DUSR       LPEX=   LTLN        ;COMMUNICATIONS EXTENSION AREA START
0016 PARU
01                      ;
02                      ; TUNING FILE DISPLACEMENTS
03                      ;
04
05          000000      .DUSR       .TUN=0              ;OFFSET TO NUMBER WORD IN PAIR
```

```
06          000001      .DUSR     .TUC=.TUN+1          ;OFFSET TO 1ST COUNT IN PAIR
07          000003      .DUSR     .TUP=.TUC+2          ;OFFSET TO 2ND COUNT OF PAIR
08          000005      .DUSR     .TUNX=.TUP+2         ;LENGTH OF COUNT PAIR
09
10          000001      .DUSR     .TUNSTK=1                        ;NUMBER STACKS IN SYSTEM
11          000002      .DUSR     .TUSTK=  .TUNSTK+.TUC-.TUN             ;STACK COUNT
12          000004      .DUSR     .TUPSTK=.TUNSTK+.TUP-.TUN              ;STACK PEND COUNT
13
14          000006      .DUSR     .TUNCEL=.TUNSTK+.TUNX            ;NUMBER CELLS IN SYSTEM
15          000007      .DUSR     .TUCEL=  .TUNCEL+.TUC-.TUN             ;CELLS COUNTS
16          000011      .DUSR     .TUPCEL=.TUNCEL+.TUP-.TUN
17
18          000013      .DUSR     .TUNBUF=.TUNCEL+.TUNX            ;BUFFERS, EXCLUDING TUNING BUFFERS
19          000014      .DUSR     .TUBUF=  .TUNBUF+.TUC-.TUN             ;COUNTS
20          000016      .DUSR     .TUPBUF=.TUNBUF+.TUP-.TUN
21
22          000020      .DUSR     .TUNOV=  .TUNBUF+.TUNX           ;OVERLAYS
23          000021      .DUSR     .TUOV=   .TUNOV+.TUC-.TUN
24          000023      .DUSR     .TUPOV=  .TUNOV+.TUP-.TUN
25
26          000025      .DUSR     TULEN=.TUNOV+.TUNX

**0000 TOTAL ERRORS, 00000 PASS 1 ERRORS

0017 PARU
ATCHA       040000      2/21
ATCON       000010      2/38
ATDIR       002000      2/36
ATLNK       010000      2/34
ATMSK       007400      2/32
ATNRS       000400      2/23
ATPAR       004000      2/35
ATPER       000002      2/26
ATRAN       000004      2/39
ATRES       001000      2/37
ATRP        100000      2/20
ATSAV       020000      2/22
ATUS1       000100      2/24
ATUS2       000040      2/25
ATWP        000001      2/27
A.SW        100000      5/05
B.SW        040000      5/06
CBKER       000323      9/23
CCANT       000320      9/20
CCAR1       000326      9/26
CCART       000325      9/25
CCKER       000305      9/09
CCLTL       000303      9/07
CCMAX       000333      9/33
CILAR       000317      9/19
CILAT       000301      9/05
CILBK       000310      9/12
CILNA       000315      9/17
CILTA       000332      9/30
CINDE       000327      9/27
CIVAR       000331      9/29
CNACM       000307      9/11
CNDBD       000302      9/06
CNEAR       000300      9/04
CNSAD       000304      9/08
CNSFS       000306      9/10
CPAR1       000330      9/28
CPARE       000324      9/24
CPHER       000312      9/14
CSFUE       000316      9/18
CSPER       000311      9/13
CSYER       000322      9/22
CTATL       000333      9/31        9/33
```

CTMAD	000314	9/16	
CTMAR	000313	9/15	
CTMLI	000321	9/21	
C.SW	020000	5/07	
DC100	000001	3/04	
DCBDK	100000	3/27	
DCC80	040000	3/25	3/35
DCCGN	000002	3/09	3/35
DCCNF	000010	3/13	3/35
DCCPO	000001	3/07	
DCCRE	004000	4/23	
DCDIO	100000	3/26	
DCEDT	100000	4/24	
DCFFO	010000	3/23	
DCFWD	004000	3/21	
DCIDI	000004	3/11	
0018 PARU			
DCKEY	000040	3/15	3/36
DCLAC	001000	3/19	3/35
DCLCD	000010	3/12	
DCLOC	000004	4/06	
DCLT8	004000	3/22	
DCLTU	020000	3/24	3/36
DCNAF	000100	3/16	
DCNI	000001	4/04	
DCPCK	000400	3/18	3/35
DCRAT	000200	3/17	
DCSPC	100000	3/29	3/35
DCSPO	002000	3/20	3/35
DCSTB	000001	3/06	
DCSTO	000001	3/08	
DCTBS	000000	2/45	
DCTIS	000002	2/47	
DCTMS	000001	2/46	
DCTO	000020	3/14	3/35
DCXON	000200	4/17	
D.SW	010000	5/08	
ENCTD	000100	13/50	
ENDOS	000020	13/51	
ENINF	000010	13/52	
ENMAP	100000	13/41	
ENMEC	010000	13/43	
ENMN3	000400	13/47	
ENMNV	002000	13/45	
ENRDO	000001	13/55	
ENRTO	000002	13/54	
ENSOS	000004	13/53	
ENUEC	020000	13/42	
ENUMN	000200	13/48	
ENUN3	001000	13/46	
ENUNV	004000	13/44	
ERABT	000110	8/16	
ERADR	000032	7/30	
ERBAD	000122	8/26	
ERBLT	000126	8/30	
ERBSP	000123	8/27	
ERCAN	000116	8/22	
ERCHA	000014	7/16	
ERCLO	000106	8/14	
ERCM3	000025	7/25	
ERCMS	000062	7/54	
ERCRE	000011	7/13	
ERCUS	000063	7/55	
ERD2S	000054	7/48	
ERDCH	000065	7/57	
ERDDE	000055	7/49	
ERDE1	000013	7/15	
ERDIO	000034	7/32	

ERDIR	000035	7/33
ERDIU	000056	7/50
ERDLE	000012	7/14
ERDNI	000066	7/58
ERDNM	000036	7/34
ERDOP	000111	8/17
ERDSN	000053	7/47
0019 PARU		
ERDTO	000101	8/09
ERENA	000102	8/10
EREOF	000006	7/10
EREXQ	000017	7/19
ERFGE	000070	7/60
ERFIL	000030	7/28
ERFIU	000060	7/52
ERFMT	000121	8/25
ERFNM	000001	7/05
ERFNO	000000	7/04
ERFOP	000015	7/17
ERFRM	000132	8/34
ERFUE	000016	7/18
ERIBS	000045	7/41
ERICB	000046	7/42
ERICD	000003	7/07
ERICM	000002	7/06
ERIDS	000052	7/46
ERINT	000130	8/32
ERLDE	000057	7/51
ERLLI	000022	7/22
ERMCA	000103	8/11
ERMDE	000100	8/08
ERMEM	000026	7/26
ERML	000036	9/34
ERMPR	000074	8/04
ERMPT	000071	8/01
ERNDD	000067	7/59
ERNIR	000114	8/20
ERNLE	000075	8/05
ERNMC	000113	8/19
ERNMD	000051	7/45
ERNOT	000042	7/38
ERNSE	000125	8/29
ERNTE	000076	8/06
ERNUL	000020	7/20
EROPD	000072	8/02
EROPM	000120	8/24
EROVA	000040	7/36
EROVF	000112	8/18
EROVN	000037	7/35
EROVR	000131	8/33
ERPAR	000024	7/24
ERQOV	000117	8/23
ERQTS	000050	7/44
ERRD	000033	7/31
ERRDY	000127	8/31
ERRPR	000007	7/11
ERRTN	000023	7/23
ERSCP	000064	7/56
ERSDE	000077	8/07
ERSDL	000105	8/13
ERSEL	000031	7/29
ERSFA	000107	8/15
ERSIM	000047	7/43
ERSOF	000134	8/36
ERSPC	000027	7/27
ERSPT	000133	8/35
0020 PARU		
ERSQF	000044	7/40

ERSRR	000104	8/12	
ERSV1	000004	7/08	
ERTID	000061	7/53	
ERTIM	000041	7/37	
ERUFT	000021	7/21	
ERUSZ	000073	8/03	
ERWPR	000010	7/12	
ERWR0	000005	7/09	
ERXMT	000043	7/39	
ERXMZ	000115	8/21	
ERZCB	000124	8/28	
E.SW	004000	5/09	
F.SW	002000	5/10	
G.SW	001000	5/11	
H.SW	000400	5/12	
I.SW	000200	5/13	
J.SW	000100	5/14	
K.SW	000040	5/15	
LCOND	000002	15/45	
LOV	000001	15/44	
LPC	000004	15/47	15/49
LPEX	000005	15/51	
LPN	000000	15/43	15/49
LTLN	000005	15/49	15/51
LTPR	000003	15/46	
L.SW	000020	5/16	
M.SW	000010	5/17	
N.SW	000004	5/18	
OVBLK	000003	15/12	
OVDIS	000002	15/10	
OVNAD	000004	15/13	
OVNDS	000000	15/05	
OVRES	000001	15/09	
O.SW	000002	5/19	
P1CBK	110012	11/40	
P1CSO	110003	11/10	
P1CUI	110010	11/38	
P1DPE	110007	11/37	
P1EMT	110022	12/01	
P1ERC	110016	11/56	
P1IDA	110004	11/16	
P1ILL	110011	11/43	
P1IPB	110014	11/48	
P1ITR	110015	11/53	
P1MDD	110005	11/21	
P1MDT	110006	11/30	
P1PAR	110017	11/58	
P1PSF	110023	12/02	
P1PSH	110013	11/45	
P1SDE	110002	11/08	
P1SPL	110021	11/60	
P2CBK	120012	11/41	
P2CSO	120003	11/11	
P2IDA	120004	11/17	
P2IPB	120014	11/49	
P2ITR	120015	11/54	
P2MDD	120005	11/22	
0021 PARU			
P2MDT	120006	11/31	
P2PSH	120013	11/46	
P3CSO	130003	11/12	
P3IDA	130004	11/18	
P3IPB	130014	11/50	
P3MDD	130005	11/23	
P3MDT	130006	11/32	
P4CSO	140003	11/13	
P4IDA	140004	11/19	
P4IPB	140014	11/51	

P4MDD	140005	11/24						
P4MDT	140006	11/33						
P5CSO	150003	11/14						
P5MDD	150005	11/25						
P5MDT	150006	11/34						
P6MDD	160005	11/26						
P6MDT	160006	11/35						
P7MDD	170005	11/29						
PNCBK	100012	9/49	11/40	11/41				
PNCSO	100003	9/43	11/10	11/11	11/12	11/13	11/14	
PNCUI	100010	9/48	11/38					
PNDPE	100007	9/47	11/37					
PNEMT	100022	9/60	12/01					
PNERC	100016	9/56	11/56					
PNIDA	100004	9/44	11/16	11/17	11/18	11/19		
PNILL	100011	9/51	11/43					
PNIPB	100014	9/53	11/48	11/49	1150	11/51		
PNITR	100015	9/55	11/53	11/54				
PNMDD	100005	9/45	11/21	11/22	11/23	11/24	11/25	11/26
		11/29						
PNMDT	100006	9/46	11/30	11/31	11/32	11/33	11/34	11/35
PNMEM	100020	9/58						
PNMPE	100001	9/41						
PNPAR	100017	9/57	11/58					
PNPSF	100023	10/02	12/02					
PNPSH	100013	9/52	11/45	11/46				
PNSDE	100002	9/42	11/08					
PNSPL	100021	9/59	11/60					
P.SW	000001	5/20						
QAC2	000012	15/32	15/34					
QCOND	000011	15/31						
QNUM	000001	15/23						
QOCH	000010	15/30						
QPC	000000	15/22	15/34					
QPEX	000013	15/35						
QPRI	000005	15/27						
QRR	000006	15/28						
QSH	000003	15/25						
QSMS	000004	15/26						
QTLN	000013	15/34	15/35					
QTLNK	000007	15/29						
QTOV	000002	15/24						
Q.SW	100000	5/21						
R.SW	040000	5/22						
SCAMX	000030	6/10						
SCAUN	000000	6/53						
SCBAD	000004	6/19						
SCBAS	000013	6/49	6/50					
0022 PARU								
SCBPB	000001	6/24	6/25	6/26				
SCCLI	000004	6/42	6/43					
SCCPL	000006	6/44	6/45					
SCDBS	000400	6/07						
SCDSK	000003	6/18						
SCEXT	000005	6/12						
SCFNL	000006	6/11						
SCFPB	000006	6/25						
SCFZW	000021	6/26	6/27					
SCGO	000002	6/55						
SCIDV	000020	6/51						
SCINS	000001	6/38						
SCINT	000003	6/41	6/42					
SCLLG	000204	6/09						
SCMAP	000011	6/23						
SCMER	000012	6/13						
SCNGO	000004	6/56						
SCNSO	000100	6/30						
SCNVW	000022	6/27						

SCOF1	000012	6/48	6/49		
SCOFA	000010	6/46	6/47		
SCPAR	000002	6/40			
SCPB1	000011	6/47	6/48		
SCPBA	000007	6/45	6/46		
SCPNM	000004	6/22	6/23	6/25	6/26
SCPPA	000006	6/17			
SCPPL	000000	6/16			
SCPSA	000002	6/39	6/40		
SCPSH	000001	6/21	6/23		
SCRRL	000100	6/08			
SCSOP	000017	6/32			
SCSTR	000016	6/14			
SCSWC	000013	6/50			
SCSYS	000000	6/20			
SCTBP	000000	6/37			
SCTIM	177660	6/15			
SCUN	000001	6/54			
SCWPB	000377	6/06			
SCZMX	000005	6/43	6/44		
SFBRK	000001	6/29			
SFINT	100000	6/28			
SNSOU	000072	6/31	6/32		
S.SW	020000	5/23			
TAC0	000001	14/07			
TAC1	000002	14/08			
TAC2	000003	14/09			
TAC3	000004	14/10			
TELN	000011	14/15			
TFP	000016	14/20			
TID	000012	14/16			
TKLAD	000014	14/18	14/24		
TLN	000015	14/24			
TLNB	000021	14/25			
TLNK	000007	14/13	15/29		
TPC	000000	14/06	14/24	14/25	
TPRST	000005	14/11	15/27		
TSABT	004000	14/34			
TSL	000017	14/21			
0023 PARU					
TSO	000020	14/22	14/25		
TSP	000015	14/19			
TSRDO	010000	14/33			
TSRSV	002000	14/35			
TSSUS	040000	14/31			
TSSYS	100000	14/30			
TSUPN	001000	14/36			
TSUSR	000400	14/37			
TSXMT	020000	14/32			
TSYS	000006	14/12			
TTICH	020040	3/36			
TTMP	000013	14/17			
TTOCH	143432	3/35			
TULEN	000025	16/26			
TUSP	000010	14/14			
T.SW	010000	5/24			
UDBAD	177770	2/12			
UDBAT	177764	2/10			
UDBBK	177766	2/13			
UDBBN	000017	2/14			
UDDL	177777	2/11			
UFCA1	000024	1/42			
UFDEL	000022	2/08			
UFEA1	000030	1/46			
UFFA1	000037	1/53			
UFLA1	000034	1/50			
UFLAD	000007	1/25			
UFLAN	000014	1/31			

UFNA1	000032	1/48					
UFPT	000030	13/35					
UFTAC	000013	1/29					
UFTAD	000012	1/28	2/12				
UFTAT	000006	1/23	2/10				
UFTBC	000011	1/27					
UFTBK	000010	1/26	2/13				
UFTBN	000041	1/58	2/14				
UFTBP	000042	1/59					
UFTCA	000025	1/43					
UFTCB	000026	1/44					
UFTCH	000043	1/60					
UFTCN	000044	2/01	2/07				
UFTDC	000022	1/40	2/10	2/11	2/12	2/13	2/14
UFTDL	000021	1/36	2/08	2/11			
UFTDR	000036	1/52					
UFTEA	000031	1/47					
UFTEL	000045	2/07					
UFTEX	000005	1/22	6/11	6/12			
UFTFA	000040	1/54					
UFTFN	000000	1/21	2/07	2/08	6/11	6/12	
UFTHM	000015	1/32					
UFTLA	000035	1/51					
UFTLK	000007	1/24					
UFTNA	000033	1/49					
UFTP1	000016	1/33					
UFTP2	000017	1/34					
UFTST	000027	1/45					
UFTUC	000020	1/35					
UFTUN	000023	1/41					
0024 PARU							
UFTYD	000014	1/30					
UST	000400	13/05					
USTAC	000015	13/24					
USTBR	000012	13/21					
USTCH	000013	13/22					
USTCS	000010	13/19					
USTCT	000014	13/23					
USTDA	000006	13/17					
USTEN	000023	13/33					
USTES	000003	13/14					
USTFC	000016	13/25					
USTHU	000007	13/18					
USTIA	000023	13/31	13/33				
USTIN	000017	13/26					
USTIT	000011	13/20					
USTNM	000004	13/15					
USTOD	000020	13/27					
USTP	000012	13/07					
USTPC	000000	13/10					
USTRV	000022	13/29					
USTSA	000005	13/16					
USTSS	000002	13/13					
USTSV	000021	13/28					
USTZM	000001	13/12					
U.SW	004000	5/25					
V.SW	002000	5/26					
W64CH	001400	4/41					
W64DC	000000	4/32					
W64DT	000001	4/37					
W64LS	000400	4/34					
W64MS	001000	4/36					
W64RT	000002	4/39					
W.SW	001000	5/27					
X.SW	000400	5/28					
Y.SW	000200	5/29					
Z.SW	000100	5/30					
.TUBU	000014	16/19					

.TUC	000001	16/06	16/07	16/11	16/15	16/19	16/23	
.TUCE	000007	16/15						
.TUN	000000	16/05	16/06	16/11	16/12	16/15	16/16	16/19
		16/20	16/23	16/24				
.TUNB	000013	16/18	16/19	16/20	16/22			
.TUNC	000006	16/14	16/15	16/16	16/18			
.TUNO	000020	16/22	16/23	16/24	16/26			
.TUNS	000001	16/10	16/11	16/12	16/14			
.TUNX	000005	16/08	16/14	16/18	16/22	16/26		
.TUOV	000021	16/23						
.TUP	000003	16/07	16/08	16/12	16/16	16/20	16/24	
.TUPB	000016	16/20						
.TUPC	000011	16/16						
.TUPO	000023	16/24						
.TUPS	000004	16/12						
.TUST	000002	16/11						

Figure B.1 Listing of PARU.LS

Appendix C

Real-time Programming Examples

This appendix contains two examples of assembly language programs written for a real-time environment.

TIMEC Program

The first example is TIMEC, a bare-bones program that creates an additional task at the same priority. During execution, TIMEC creates TASK at the same priority as itself (0). The new task competes for CPU control, gets it when TIMEC suspends itself, and retains it until it suspends itself. Each task prints a message on the console when it gains control. TIMEC suspends itself for two seconds, and TASK suspends itself for four seconds. After roughly eight seconds elapse, the console shows the following messages:

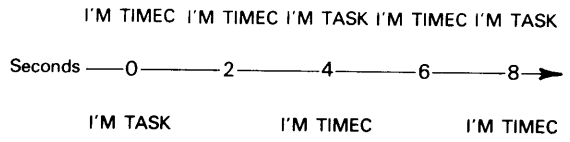
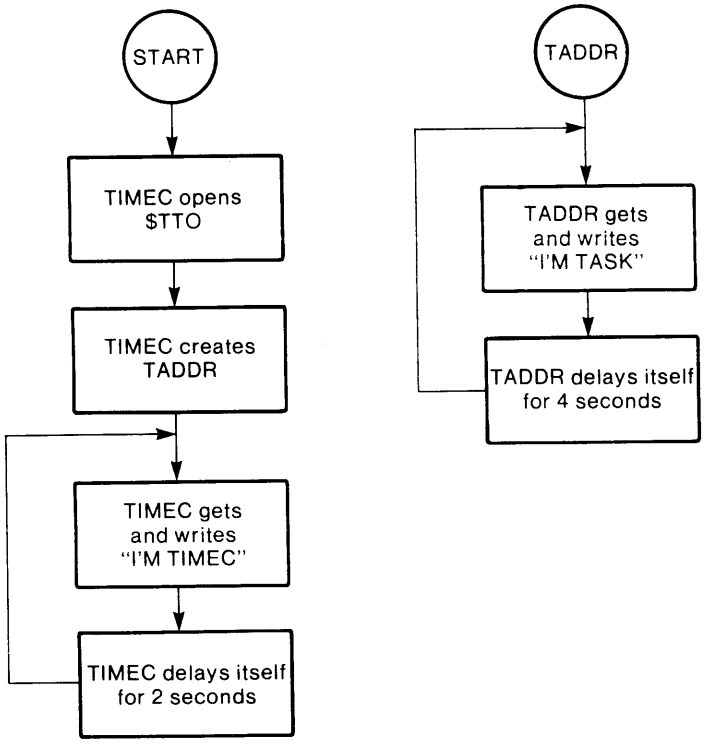


Figure C.1 TIMEC and TASK messages ID-00497

I'M TIMEC
I'M TASK
I'M TIMEC
I'M TASK
I'M TIMEC
I'M TIMEC
I'M TASK
I'M TIMEC

The messages appear in syncopated fashion because the tasks suspend themselves for different times, as shown in Figure C.1. A flowchart of the TIMEC program appears in Figure C.2, and Figure C.3 lists the program code.

Because TIMEC includes no code to return to the CLI, you must use the RDOS interrupts CTRL-A or CTRL-C to stop it and return to the CLI.



Each procedure box represents a request to the system, which then surrenders control to the task scheduler.

Figure C.2 TIMEC flow chart

SD-00572

```
                    .TITL TIMEC
                    .COMM TASK,2*400+1
                    .EXTN .TASK
                    .ENT START
                    .TXTM 1
                    .NREL

START:              LDA 0, NTTO             ;Pointer to console
                                            ;output filename.
                    SUB 1,1                 ;Use default mask
                                            ;on $TTO.
                    .SYSTM
                    .OPEN 0                 ;Open $TTO on channel 0.
                    JMP ERROR               ;On most errors, let the
                                            ;CLI explain.
                    SUB 0,0                 ;Give new task priority
                                            ;and ID of 0.
                    LDA 1,.TADDR            ;Start task at this
                                            ;address.
                    .TASK                   ;Create the task.
                    JMP ERROR
TIMEC:              LDA 0, .TIMES           ;TIMEC, pick up
                                            ;pointer to message.
                    .SYSTM
                    .WRL 0                  ;Write message.
                    .JMP ERROR
                    LDA 1, .S2              ;Pointer to interval.
                    .SYSTM
                    .DELAY                  ;TIMEC, delay
                    JMP ERROR               ;yourself, giving TASK
                    JMP TMEC                ;control until delay
                                            ;expires.

TADDR:              LDA 0, .T2MES           ;TASK, pick up
                                            ;pointer to message.
                    .SYSTM
                    .WRL 0                  ;Write message.
                    JMP ERROR
                    LDA 1, .S4              ;Pointer to interval.
                    .SYSTM                  ;Delay yourself,
                    .DELAY                  ;giving TIMEC control.
                    JMP ERROR
                    JMP TADDR               ;When you awaken, write
                                            ;message again.
NTTO:               .+1*2
                    .TXT "$TTO"
.TADDR:             TADDR

.TIMES:             .+1*2
                    .TXT "I'M TIMEC.(15)"
T2MES:              .+1*2
                    .TXT "I'M TASK.(15)"
.S2:                20.                     ;20*10 Hz RTC frequency
                                            ;is 2 seconds.
.S4:                40                      ;40*10Hz is 4 seconds.

ERROR:              .SYSTM
                    .ERTN
                    JMP ERROR               ;Reserved, never taken.
                    .END START
```

Figure C.3 TIMEC program listing

EXAMPLE Program

The second program, EXAMPLE, is a multitasking program that uses overlays; it shows multitask overlay calls and a queued overlay task. Figure C.4 charts the program flow. The assembler listings for EXAMPLE and its two overlays, QUE and COMP, appear in Figure C.5. In EXAMPLE, the main program task opens the console input, output, and overlay files; then it sets its priority to 40_8 and creates a second task via call .QTSK at priority 30_8. The new task, called QUE, will be created and readied every three seconds. After creating task QUE, the main program momentarily retains CPU control and types a prompt (?) on the system console. The main program task recognizes two commands: the letter B, meaning return to the CLI, and the letter C, meaning load overlay COMP and execute the code within it.

Overlay COMP types the message:

I AM A DATA GENERAL COMPUTER.

COMP then releases the overlay node and returns to the prompt loop in the main program. (On characters other than B or C, the main program repeats the prompt loop.)

Shortly after the main program has typed its prompt, and while it is waiting for input, the QUE task is readied. At the next device interrupt (from the real time clock, console, etc.), rescheduling occurs and the task scheduler gives QUE control of the CPU because it has a higher priority than the main program. The system, under direction of the task scheduler, suspends the main program, loads the overlay containing QUE, and transfers control to code in QUE. QUE then types the message:

I'M THE QUEUED TASK...ABOUT TO .OVKIL MYSELF.

At this point, QUE prints the prompt (?) and kills itself via the call .OVKIL command. Control returns to the main program, which again waits for input. In three seconds, task QUE is recreated and readied, and the entire sequence repeats itself.

When QUE is ready to run, it gains control, types its message, and kills itself very quickly. In fact, because QUE issues system calls, it is briefly suspended before it can type the message and prompt—thus allowing the main program a slice of CPU control. This scheme enables the person who runs the program to type commands B or C at any time and receive a very fast response.

Both tasks (the main program and QUE) are completely unaware of one another. Furthermore, when an interrupt occurs and the scheduler decides to suspend one task and execute another, the original task simply continues from the point at which it was suspended—which can be from any location in its address space. When QUE kills itself, its entire state (TCB data) is wiped out; after the .QTSK interval, it is created as a brand-new task. Thus, there is no simple way that QUE can return control to the prompt loop in the main program. This is why QUE is coded to type a prompt before killing itself. (The .XMT and .REC commands could return control to the main prompt loop, but this would produce a far more complex example.)

A dialogue with the EXAMPLE program might transpire as follows:

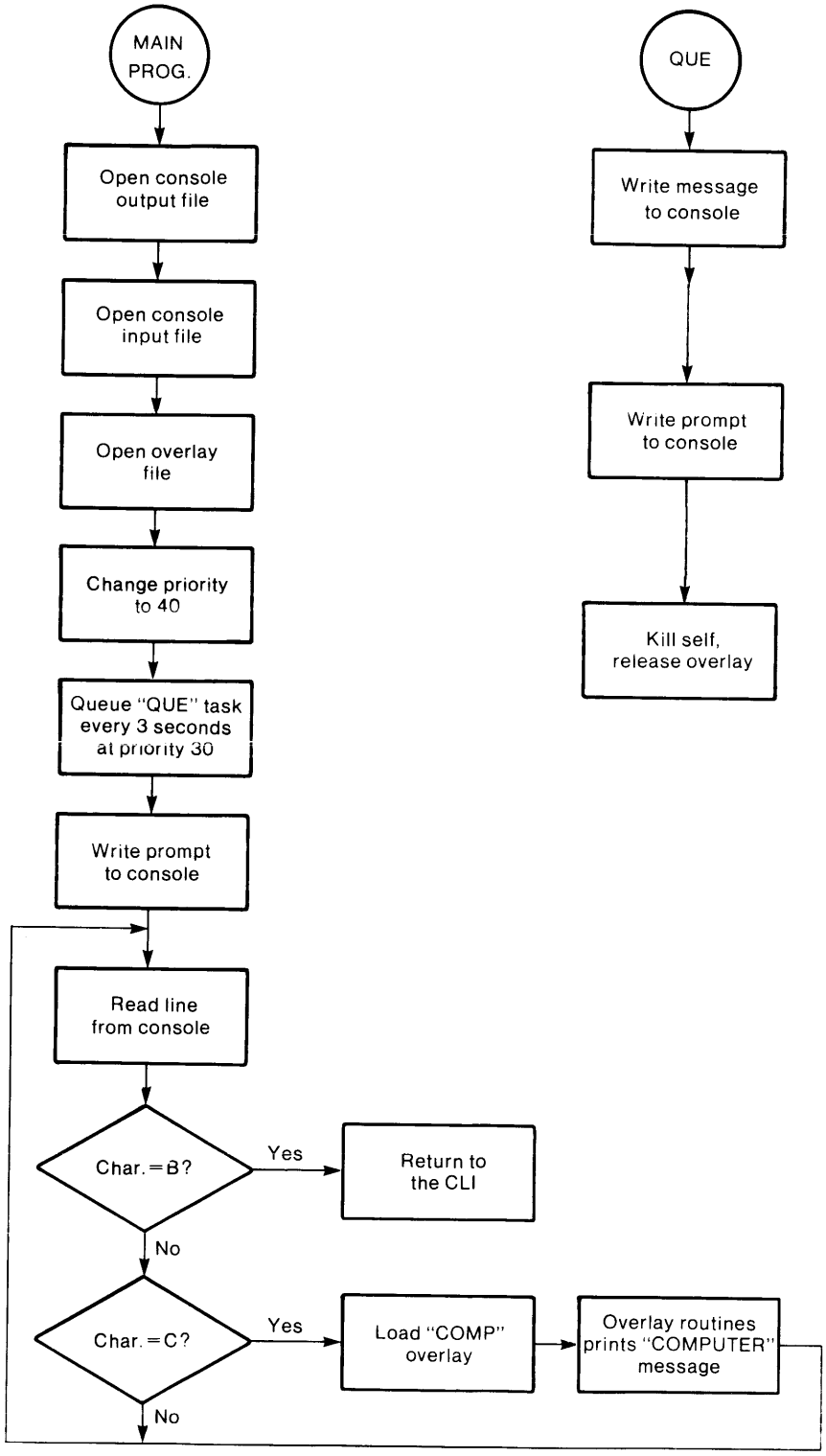
```
R
EXAMPLE ⟨CR⟩

?
I'M THE QUEUED TASK...ABOUT TO .OVKIL MYSELF.

?
C ⟨CR⟩
I'M A DATA GENERAL COMPUTER.

?
I'M THE QUEUED TASK...ABOUT TO .OVKIL MYSELF.
B ⟨CR⟩

R
```

The assembler command for the EXAMPLE program was:

```
MAC/L (EXAMPLE,QUE,COMP) ⟨CR⟩
```

The load line was:

```
RLDR 2/K EXAMPLE [QUE,COMP] ⟨CR⟩
```



As with TIMEC, each procedure box represents a request
a request to the system, which then surrenders full control
to the task scheduler.

Figure C.4 EXAMPLE flowchart

SD-00573

```
                        .TITLE    EXAMPLE
02                      .ENT      AGAIN ICOMP,IQUE,ERROR      ;OVERLAYS NEED THESE
03                      .EXTN     OCOMP,OQUE,COMP,QUE         ;OVERLAYS CONTAIN THESE.
04                      .EXTN     .PHI,.QTSK,.TOVLD           ;GET TASK CODE FROM SYS.LB.
05 000001               .TXTM 1                               ;PACK BYTES LEFT TO RIGHT.
06
07      ;For RDOS revisions 6.00 through 6.20, apply patch "JMP .+2" to
08      ;location "Q.TSK+333" of any save file that uses .GTSK.
09      ;Include the debugger (RLDR/D) or symbol table (.EXTN .SYM.)
10      ;to patch with the SEDIT editor.
11
12              .ZREL
13 00000-177400 PMASK:   .177400                              ;MASK FOR FIRST 2 BYTES IN LINE BUFFER.
14 00001-000226'OCHAIN:  OVCHN                                ;POINTER TO CHANNEL NUMBER OF OVLY FILE.
15 00002-002003-ERROR:   JMP      @.+1                        ;ON ERROR, JUMP TO
16 00003-000213'         SERR                                 ;     ERROR HANDLER SERR.
17
18                      .NREL
10
20              ; OPEN CONSOLE OUTPUT, INUT, AND OVERLAY FILES FOR I/O.
21
22 0000'020445 START     LDA      0,NTTO                      ;BYTE POINTER TO CONSOLE OUTPUT FILENAME.
23                                                            ;(FOR OPERATION IN EITHER GROUND, INCLUDE
24                                                            ;.GCOUT, .GCTN CALLS BEFORE OPEN CALLS.)
25 00001'126400          SUB 1,1                              ;SET DEFAULT DEVICE CHARACTERISTIC MASK.
26 00002'006017          .SYSTM                               ;OPEN THE CONSOLE OUTPUT FILE
27 00003'014000          .OPEN    0                           ; ON CHANNEL 0.
28 00004'004002-           JSR    ERROR                       ;CAPTURE ANY ERROR. (JSR HELPS DEBUG.)
29
30 00005'020444          LDA      0, NTTI                     ;BIT POINTER TO CONSOLE INPUT FILENAME.
31 00006'006017          .SYSTM                               ;OPEN CONSOLE INPUT FILE ON
32 00007'014001          .OPEN    1                           ; CHANNEL 1. (AC1 STILL CONTAINS MASK 0.)
33 00010'004002-           JSR    ERROR                       ; ERROR.
34
35 00011'020444          LDA      0,OFILE                     ; GET OVERLAY FILENAME.
36 00012'032001-         LDA      2,@OCHAN                    ; GET CHANNEL NUMBER FOR OVERLAY FILE.
37 00013'006017          .SYSTM                               ; OPEN OVERLAY FILE ON THE
38 00014'012077          .OVOPN   77                          ; SPECIFIED CHANNEL.
39 00015'004002-           JSR    ERROR                       ; ERROR.
40
41              ;PROCEED--SET YOUR PRIORITY TO 40 AND QUEUE A TASK.
42
43 00016'020446          LDA      0,C40                       ;GET A 40.
44 00017'077777          .PRI                                 ; SET YOUR PRIORITY TO 40.
45
46 00020'030556          LDA      2,QADDR                     ;GET TASK QUEUE TABLE ADDR.
47 00021'077777          .QTSK                                ;SET UP OVLY TASK TO RUN EVERY 3 SECONDS.
48 00022'004002-           JSR    ERROR                       ; ERROR.
49
50              ;THIS IS THE MAIN PROMPT AND KEYBOARD LISTENER LOOP.
51
52 00023'020442          AGAIN:   LDA 0,PROMPT                ;BYTE POINTER TO PROMPT.
53 00024'006017                   .SYSTM                     ;WRITE THE PROMPT
54 00025'017000                   .WRL 0                     ; TO THE CONSOLE ON CHANNEL 0.
55 00026'004002-                  JSR ERROR                  ; ERROR.
56 00027'020441                   LDA 0,LINER                ;BYTE POINTER TO LINE BUFFER.
57 00030'006017                   .SYSTM                     ;READ A LINE FROM
58 00031'015401                   .RDL 1                     ; CONSOLE KEYBOARD ON CHANNEL 1.
59 00032'004002-                  JSR ERROR                  ; ERROR.

01              ; CHECK LINE FOR B OR C. (THIS MIGHT BE STREAMLINED FOR A COMPUTER
02              ; WITH HARDWARE LOAD, STORE BYTE.)
03
```

Figure C.5 EXAMPLE program listing

```
04 0033'024436              LDA     1,LINE          ;GET RIGHT WORD (2 CHARS) FROM LINE BUFFER.
05 00034'03000-             LDA     2,PMASK         ;MASK TO STRIP PARITY, RIGHT CHAR IN AC2.
06 00035'133700             ANDS    1,2             ;ISOLATE FIRST CHAR IN BITS 0-6 OF AC2,SWAP.
07 00036'023536             LDA     1,B             ;GET A "B".
08 00037'147415             SUB     2,1,SNR         ;SKIP IF FIRST CHAR WASN'T A "B".
09 00040'000550               JMP   BYE             ; ON "B", RETURN TO THE CLI.
10 00041'024534             LDA     1,C             ;GET A "C".
11 00042'146415             SUB#    2,1,SNR         ;SKIP IF FIRST CHAR WASN'T A "C".
12 00043'000514               JMP   GCOMP           ; ON "C", GO TO THE "COMPUTER" OVERLAY.
13 00044'000757             JMP     AGAIN           ;NOT "R" OR "C", IGNORE CHARACTER, TRY AGAIN.
14
15
16 00045'000114"NTIO:       .+1*2                   ;POINT TO
17 00046'022124             .TXT    "$TTO"          ; FILENAME "$TTO".
18         052117
19         000000
20
21 00051'000124"NTTI:       .+1*2                   ;POINT TO
22 00052'022124             .TXT    "$TTI"          ; FILENAME "$TTI".
23         052111
24         000000
25
26 00055'000134"OFILE:      .+1*2                   ;POINT TO
27 00056'042530             .TXT    "EXAMPLE.OL"    ; OVERLAY FILENAME.
28         040515
29         050114
30         042456
31         047514
32         000000
33
34 00064'000040 C40:        40                      ;NEW PRIORITY FOR MAIN PROGRAM TASK.
35
36 00065'000154"PROMPT:     .+1*2                   ;POINT TO
37 00066'037415             .TXT    "?(15)"         ; MAIN PROGRAM PROMP.
38         000000
39
40 00070'000162"LINEP:      LINE*2                  ; POINTER TO FIRST BYTE OF LINE BUFFER.
41 00071'000103 LINE:       .BLK    132./2*1        ; BUFFER TO HOLD MAX. LINE LENGTH.
42
43 00174'000102 B:          "B                      ;ASCII "B".
44 00175'000103 C:          "C                      ;ASCII "C".
45 00176'000216'QADDH:      QTAR                    ;ADDRESS OF "QUE" TASK QUEUE TABLE.
46
47
48          ; THIS CODE PROCESSES THE "C" CHARACTER. IT LOADS THE "COMP"
49          ; OVERLAY AND TRANSFERS TO WRITE-LINE CODE IN THE OVERLAY.
50
51 00177'020410 GCOMP:      LDA     0,ICOMP         ;GET "COMPUTER" OVERLAY NAME.
52 00200'126400             SUB     1,1             ;SPECIFY CONDITIONAL LOADING.
53 00201'032001-            LDA     2,@OCHAN        ; GET OVERLAY FILE CHANNEL NUMBER.
54 00202'077777             .TOVLD                  ; REQUEST SYSTEM ACTION.
55 00203'004002-              JSR   ERROR           ; ERROR.
56 00204'006402             JSR     @ACOMP          ;EXECUTE THE OVERLAY CODE, THEN
57 00205'000616             JSR     AGAIN           ; GO BACK FOR MORE INPUT.
58
59 00206'077777 ACOMP:      COMP                    ;START ADDRESS IN OVERLAY.
60 00207'077777 ICOMP:      OCOMP                   ;"COMPUTER" OVERLAY IDENTIFIER.


01
02
03              ; THIS CODE PROCESSES THE "B" CHARACTER. IT TERMINATES
04              ; THE PROGRAM AND RETURNS TO THE CLI.
05
```

Figure C.5 EXAMPLE program listing (continued)

```
06 00210'006017 BYE:        .SYSTM                          ;RETURN TO THE RDOS CLI.
07 00211'004400             .RTN                            ;
08 00212'000002-            JMP         ERROR               ;RESERVED, NEVER TAKEN.
10
11              ; THIS IS THE ERROR HANDLER.
12
13 00213'006017 SERR:       .SYSTM                          ;LET THE CLI REPORT WHAT'S WRONG.
14 00214'006400             .ERTN                           ;
15 00215'000776             JMP                             ;NEVER TAKEN.
16
17
18              ; THIS IS THE QUEUE TABLE FOR THE "QUE" OVERLAY TASK.
19
20 00216'077777 QTAB:       QUE                             ;STARTING ADDRESS FOR THE TASK.
21 00217'177777             -1                              ;EXECUTE UNLIMITED NUMBER OF TIMES.
22 00220'077777 IQUE:       OQUE                            ;OVERLAY IDENTIFIER -- .ENTO.
23 00221'177777             -1                              ;STARTING HOUR: RIGHT NOW.
24 00222'000001             .BLK        1                   ;STARTING SECOND (UNIMPORTANT HERE).
25 00223'000430             1B7+30                          ;TASK ID OF 1, PRIORITY OF 30.
26 00224'000003             3.                              ;RERUN EVERY 3 SECONDS.
27 00225'000001             .BLK        1                   ;SYSTEM WORD.
28 00226'000002 OVCHN:      2                               ;USE CHANNEL 2 FOR THE OVERLAY FILE.
29 00227'000000             0                               ;CONDITIONAL OVERLAY LOADING.
30 00230'000001             .BLK        1                   ;SYSTEM WORD.
31 00231'000001             .BLK        1                   ;WORD FOR EXTENDED QUEUE TABLE USAGE.
32
33                          .END        START               ;STARING ADDRESS IS START.

**00000 TOTAL ERRORS, 000000 PASS 1 ERRORS

02                          .TITLE      QUE
03                          .ENT        QUE
04                          .ENTO       OQUE
05                          .EXTN       ERROR,IQUE,AGAIN
06          000001          .TXTM       1
07                          .NREL
08
09              ; "QUE" OVERLAY - WRITES MESSAGE TO CONSOLE, KILLS SELF AND
10              ; QUEUED TASK.
11
12 00000'020420 QUE:        LDA         0,MESS                  ; BYTE POINTER TO MESSAGE.
13 00001'006017             .SYSTM                          ; WRITE MESSAGE
14 00002'017000             .WRL        0                   ;  TO CONSOLE OUT.
15 00003'006411               JSR       @ERR                ; ERROR RETURN.
16 00004'020411             LDA         0,PROMPT                ; BYTE POINTER TO PROMPT.
17 00005'006017             .SYSTM                          ; WRITE PROMPT
18 00006'017000             .WRL        0                   ;  TO CONSOLE OUT (FOR CONSISTENCY).
19 00007'006405             JSR         @ERR                ;  ERROR.
20
21 00010'022403             LDA         0,@OQ               ; GET THE OVERLAY IDENTIFIER.
22 00011'077777             .OVKIL                          ; RELEASE OVERLAY AND KILL TASK.
23 00012'006402               JSR       @ERR                ;  ERROR.
24
25 00013'077777 OQ:         IQUE                            ; OVERLAY IDENTIFIER.
26 00014'077777 ERR:        ERROR                           ; ERROR HANDLER.
27
28 00015'000034"PROMPT:     .+1*2                           ; POINT TO
29 00016'037415             .TXT        "?(15)"             ;  PROMPT.
30          000000
31
32 00020'000042"MESS:       .+1*2                           ; POINT TO "QUEUED" MESSAGE.
33 00021'044447             .TXT        "I'M THE QUEUED TASK...READY TO .OVKIL MYSELF.(15)"
34          066440
35          072150
```

Figure C.5 EXAMPLE program listing (continued)

```
36      062440
37      070565
38      062565
39      062544
40      020164
41      060563
42      065456
43      027056
44      071145
45      060544
46      074440
47      072157
48      020117
49      053113
50      044514
51      020155
52      074536
53      062554
54      063056
55      006400
56
57                      .END

**00000 TOTAL ERRORS, 00000 PASS 1 ERRORS

02                      .TITLE  COMP
03                      .ENT    COMP
04                      .ENTO   OCOMP
05                      .EXTN   .OVEX
06      000001          .TXTM   1
07                      .NREL
08
09              ; 'COMPUTER' OVERLAY - PRINT MESSAGE AND RETURN.
10
11 00000'054016 COMP:   STA     3,USP                   ;FOR REENTRANCY.
12 00001'020412         LDA     0,CMESS                 ;GET MESSAGE ADDR.
13 00002'006017         .SYSTM                          ;WRITE IT
14 00003'017000         .WRL    0                       ;   TO THE CONSOLE.
15 00004'006405           JSR   @ERR                    ;ERROR RETURN.
16
17 00005'022405         LDA     0,@OCP                  ;GET THE OVERLAY IDENTIFIER.
18 00006'030016         LDA     2,USP                   ; AND THE RETURN ADDRESS
19 00007'077777         .OVEX                           ;TO EXIT AND RELEASE THIS OVERLAY.
20 00010'006401           JSR   @ERR                    ;ERROR
21
22 00011'077777 ERR:    ERROR                           ;ERROR HANDLER.
23 00012'077777 OCP:    ICOMP                           ;OVERLAY IDENTIFIER.
24
25 00013'000030"CMESS:  .+1*2                           ;POINT TO "COMPUTER" MESSAGE.
26 00014'044440         .TXT    "I AM A DATA GENERAL COMPUTER.(15)"
27      060555
28      020141
29      020104
30      060564
31      060440
32      043545
33      067145
34      071141
35      066040
36      061557
37      066560
38      072564
39      062562
```

Figure C.5 EXAMPLE program listing (continued)

```
40      027015
41      000000
42
43                              .END

**00000 TOTAL ERRORS, 00000 PASS 1 ERRORS
```

Figure C.5 EXAMPLE program listing (continued)

Appendix D

Overlay Directory Structure

When you load a program that has an associated overlay file, the loader program creates an overlay directory for it. During program execution, this directory occupies low NREL memory, right above the TCB pool, and contains a four-word descriptor for each overlay. In a mapped system, the directory must fit into the lowest, 1K-block of memory.

You, or your program, can examine the overlay directory through entry USTOD in the user status table. USTOD points to the directory base; it contains -1 if there are no overlays. The overlay directory built for each multitask program has the structure shown in Figure D.1.

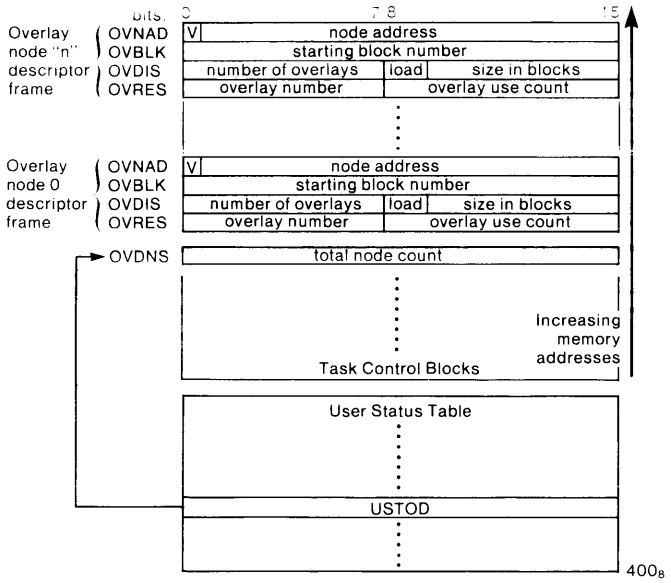


Figure D.1 Overlay directory structure (multitask) SD-00532

As Figure D.1 shows, each overlay node in the save file has a corresponding, four-word descriptor frame in the overlay directory. Bits 0 through 7 of OVRES contains the number of the overlay that currently resides in the overlay node or which RDOS is loading into it. The overlay use count (OUC) (bits 8 through 15 of OVRES) describes the number of tasks using or requesting the resident overlay. RDOS uses OUC only in a multitask environment, as explained in Chapter 5.

Bits 0 to 7 of OVIDS describe the number of overlays associated with this overlay node (ie, included in the same pair of square brackets in the RLDR command line. RDOS uses the load bit, bit 8, in multitask programs (.TOVLD). Bits 9 to 15 of this word describe the size (in integer multiples of 400_8 words, the size of each disk block) of this overlay node. OVBLK contains the starting, logical disk block address of this node's segment in the overlay file, and OVNAD contains the memory address for the start of this overlay mode. For a virtual overlay node, RDOS sets B0 of OVNAD to one.

The overlay directory built for a single-task environment is identical to that described here except that the system ignores the load bit. A program can define a maximum of 256 overlay nodes in both single- and multitask environments. The maximum number of 256-word overlay nodes is 124 (which need about 60K bytes of memory). Page zero and task scheduler space requirements limit the maximum size of a single overlay to 126 disk blocks (64K bytes).

Appendix E

Exceptional System Status

Certain serious error conditions can either halt the entire system in a crash, or cause the system to suspend processing and display an exceptional status or trap message. The message returned from exceptional status or a trap helps identify the error; no information is returned from a crash. Both exceptional status and crash conditions require full initialization of all disks that were initialized when the condition occurred.

Traps

A trap is less serious than an exceptional status or a crash; they are described here, however, because they do stop program execution. On a trap, the system displays the contents of the program counter and the accumulators on the console in this format:

TRAP (PC) (AC0)(AC1)(AC2)(AC3)

Bit 0 of the PC is carry. In both mapped and unmapped systems, a trap usually results from a violation of map protection. The memory-image file (F)BREAK.SV is created and placed in the current directory.

In its discussion of dual programming in mapped systems, Chapter 6 provides details on certain user-caused traps. Among the most common of these causes are:

- An attempt to access memory outside your logical space
- An attempt to modify write-protected memory
- More than 16 indirect references to an address
- An attempt to access a system device without having issued the .DEBL command.

Exceptional Status

In exceptional status, the system outputs the contents of the accumulators and an error code on the console, for example:

000015	*177777*	*000011*	*037500*	*100010*
AC0	AC1	AC2	AC3	Error Code

Note that if a system error caused the exceptional status, bit 0 of the error code will be reset to 0, and the rest of the code word will contain a system error number (explained in Appendix A). The dump procedure described later in this section applies to both kinds of error.

If bit 0 of the error code is set to 1, the last two digits of the error code have the following meanings:

1 File system inconsistency detected, that is, RDOS tried to return a master device block which had no record in MAP.DR.

2 RDOS detected a SYS.DR error while accessing a directory on the master device. This means that either the entry count in a block of the directory exceeds 16_8, or a free entry in the block was indicated but RDOS could not find it. If AC0 contains 16, AC2 contains the illegal count; otherwise RDOS expected a free entry but did not find it.

3 Interrupt stack overflow. The low-order bits of AC0 contain the address of the overflowed stack. If this is a system stack address (see load map), the cause can be a system device. If the address is not a system stack address, the cause is a software stack fault.

4 Inconsistent system data, such as an illegal device address. This also occurs if you issue INIT or DIR to a new disk before fully initializing it with INIT/F.

5 Master device data error; run a disk reliability test.

6 Master device timeout. If there are no obvious errors, run a disk reliability test.

7 Illegal device address on the moving-head master device. This can be caused by a misreading of the disk. Run a disk reliability test.

10 RDOS has detected an undefined interrupt and cannot clear it via an NIOC. The right byte of AC2 contains the code of the device.

12 There are not enough contiguous disk blocks available to build push space indexes.

13 Attempted RTN from level 0 in the background. Remove this instruction from the level 0 program, or execute it at a lower level.

14 Inconsistent IPB data. Perform an IPB reliability test. AC2 can give a clue to the problem, *if the following conditions were true when the exceptional status occurred:*

- Both processors were up and running the same revision of RDOS.
- No user program issued I/O commands to the IPB or overwrote the (unmapped) system.

If both conditions are true and AC2 contains -1 or a DCB address, the exceptional system status indicates an internal system (software) bug.

If AC2 has a cell address, RDOS has received an invalid message type. AC1 has the type byte. This problem indicates IPB hardware failure.

If AC2 has an address in the IPB interrupt handler (between IPBDC and IVTINT), this is the address at which the exceptional system status actually occurred. If AC0 and AC1 do not contain 64400_8, the interrupt handler detected an invalid condition such as incorrect message length. This indicates an IPB hardware failure. If AC0 and AC1 equal 64400_8, one processor timed out to the other processor, but resumed communication without booting. This would occur if the operator pressed the STOP switch and more than one-and-a-half seconds later pressed CONTINUE; or if a user program turned interrupts off for more than one-and-a-half seconds (eg, via the interrupt-disable debugger).

15 A hardware map violation (trap) occurred while a user interrupt routine or user clock had control. The AC0 data field output on the console will contain the PC, not the contents of AC0, in this exceptional status.

16 ECLIPSEs with ERCC option only. Multibit ERCC memory error. Consult the appropriate Technical Reference for your computer.

17 NOVA 3s with hardware parity option only. Hardware parity error.

20 INFOS systems only. Insufficient memory available at initialization time.

21 The spooler detected a MAP.DR error.

Controlling Exceptional Status

If you have an unmapped system, you can write your own routine to handle exceptional status situations. Your programs must store the address of your routine in location 11_8 at runtime, and restore the original value before the program ends. Your routine will then gain control at an exceptional status: the console will not display the accumulators and error code, but AC0, AC1, and AC2 will retain the contents they held at the error, and AC3 will contain the address of the error code.

If yours is a mapped system, you must modify the operating system at source level to insert your own exceptional status routine.

Producing a Core Dump

If you select the core dump feature during system generation, you can dump a core image of address space after a system crash or an exceptional status. A SYSGEN question asks if the core dump facility is desired and where the dump should go:

CORE DUMP? (0 = NO, 1 = LPT, 2 = MTA, 3 = 6030, 4 = 6097)

If you answer 0, no core dump routine is included in the operating system you are generating. If you answer 1, the line printer routine is included and dumps will go to the primary line printer, $LPT. Answering 2 causes the routine for magnetic tape to be included, and SYSGEN will ask for the number of the magnetic tape unit to receive the dump. You may specify any unit (0 through 7) on the primary controller. Answering 3 will include the routine for single-density diskette (6030), and 4, the routine for double-density diskette (6097).

An answer of 3 or 4 prompts another SYSGEN question to determine whether the dump should go to the primary or secondary controller. Single-density diskette dumps go to the third unit on the selected controller (DP3 for the primary controller, DP7 for the secondary). For double-density diskette dumps, SYSGEN asks the unit number of the device that will receive the core dump.

Note that a device need not be generated to receive core dumps, and that any diskette used to receive a core dump must be hardware formatted.

The advantages of dumping to magnetic tape or diskette are that these media are more easily shipped to Data General for analysis. Moreover, once there, the dumps can easily be duplicated to facilitate distribution to various areas for investigation of the problem.

To produce a core dump after an exceptional status, follow the instructions for the device that will receive it. After a system crash, the console will display nothing. To prepare for a core dump, if your computer's front panel has data switches, read the section called Data Switches. If your computer has a virtual console, read the section called Virtual Console.

Data Switches

1. Press the STOP switch on the front panel of the CPU.

2. Record the contents of the accumulators, the PC, carry, USER MODE, and the state of ION.

3. Lift the RESET switch.

4. Enter 11_8 in the data switches.

5. Lift the EXAMINE switch and note the number returned in the data lights.

6. Set the CPU front panel switches to the number found in location 11_8.

7. Deposit the contents of location 11_8 into AC3.

8. Push the CONTINUE switch. RDOS displays the contents of the accumulators and carry.

9. Continue by responding to questions about the device you have chosen to receive the core dump. Questions will vary, depending upon how you did your sysgen.

Virtual Console

1. Enter the Virtual Console. (Refer to the *Programmer's Reference Guide* for your CPU for specifics on how to do this.)

2. At the (!) prompt, type: 11/ ⟨NEW LINE⟩.

   RDOS displays two sets of numbers, one set beside the slash and the other after a space, at the right. For example, if you type 11/ ⟨NEW LINE⟩ your console displays numbers similar to the following:

   ```
   !11/000011 025173
   !
   ```

3. Type 3A without typing NEW LINE.

   RDOS displays the contents of accumulator three. Whatever you type at this point goes into accumulator three.

4. Type the right-hand number that displayed when you typed 11/. In our example, the number is 25173 (Note that you do not need to include leading zeros). Then type NEW LINE to put the number into AC3. Your console displays numbers similar to the following:

   ```
   !3A 0003A 25173
   !
   ```

   This means that RDOS will start running at address 25173.

5. Next, to execute the panic routine, (not a true panic but one forced by these procedures), type 25173R (without a NEW LINE).

   RDOS displays the panic code and the contents of all of the accumulators.

6. To proceed with the core dump, type P (without a NEW LINE).

7. Continue by responding to questions that RDOS displays about the device that you have chosen to receive the core dump. The questions will vary depending upon how you did your sysgen.

Line Printer Dump

In this dump, you can select portions of memory, or dump all of memory. The line printer dump has three parts: the left column shows a memory address; the middle eight columns show the contents of each word in the address; and the right column shows the ASCII value, if any, of each byte in the address. Figure E.1 illustrates a sample line printer dump.

To dump the entire address space of either a mapped or unmapped machine, press CONTINUE twice. To dump selected portions of address space, follow one of these procedures:

Unmapped Machines Load the desired starting dump address into the data switches. Press CONTINUE; the CPU will halt. Load the desired ending address of the dump into the data switches, and again press CONTINUE. You can enter as many starting/ending address pairs as you wish.

Mapped Machines RDOS will shift three bits to the left of each address that you input via the data switches, so that you can dump the full range of possible mapped addresses. That is, if data switch 15 is up and the rest down, RDOS interprets this as address 10_8, adding an implicit zero to any address you enter. To dump a range of mapped addresses, load the desired starting dump address into the data switches, and press CONTINUE. The CPU will halt. Load the desired ending address of the dump into the data switches, and again press CONTINUE. RDOS will dump all locations from the low-order address (times 10_8) to (but not including) the high-order address (times 10_8). That is, if you select low address *l* and high address *h* on the data switches, RDOS dumps locations 10_8 *1 through (10_8 *h) -1. Repeat the dumping process as often as you wish.

You can abort the core dump any time by striking any key on the console and proceed with another dump sequence as you desire.

01020	060227	014510	060277	014506	060277	014504	060277	014502	***H***H***D***B
01030	060277	014500	060277	014476	060277	040532	044532	040432	***@***>**AZIZQZ
01040	054532	176660	054524	024466	125224	000420	034012	020742	YZ**YT)6****B*!*
01050	163000	042741	025415	021414	101005	045414	020454	025405	**E***#***K*!,**
01060	106414	000404	025406	041406	045405	034012	025405	044546	******C*K*B***I*
01070	030436	061405	025377	044017	025414	044537	030431	051414	1*S***H***I*1*S*
01000	060177	024016	045010	024426	044505	020536	040422	040422	**(*J*)*IE!*A*A*
01110	034502	152120	021420	043410	175400	153102	000774	137000	98*P#*G****B****
01120	025776	044511	014467	000410	176440	002466	005731	005756	+*II*7*** *6****
01130	000011	000011	001014	015766	000406	011766	030453	004434	************1***
01140	004404	000763	034450	000535	054455	006447	126400	044450	****9(**Y-*1**I(
01150	006446	024446	034475	137000	025400	006443	024441	010440	*&)&9******#)!*
01160	102120	107037	125401	002436	006423	000763	000000	000013	*P**************
01170	006326	000005	001014	054411	006420	020411	143000	006426	******Y***!*****
01200	006407	006414	005124	005124	000000	002014	041057	003146	*****T******B/**
01210	001400	177777	003777	002076	002013	002617	003137	000000	*******>********
01220	001631	000000	002001	100000	002031	000000	006000	003257	****************
01230	002000	000405	002032	000000	000424	000714	002367	177770	****************

Figure E.1 Sample line printer dump

Magnetic Tape Dump

To dump to magnetic tape, follow these steps:

1. Select the same unit number specified to SYSGEN on a magnetic tape drive, and make sure no other drive has this number. Mount a blank tape (300 feet or more), with write-enable ring inserted, on this drive. Then press drive switches LOAD and ON LINE.

2. Press the CPU switch CONTINUE. The dump program then displays the message, READY?

3. Again, press the CPU switch CONTINUE. The dump program copies all memory addresses to the tape and then displays the message DONE, followed by READY, on the console. To stop the program, press CPU switch STOP. To produce another dump, press the RESET and UNLOAD switches on the tape drive; mount another tape; and repeat steps two and three.

4. If you have forgotten a step, the program dislays the message ERROR, then READY? Execute the step and press the CPU switch CONTINUE.

Core dumps from machines with the maximum amount of memory supported by RDOS (512 KB) will fit onto one tape. The magnetic tape cannot be read or copied under RDOS. You may, however, write to file numbers one and higher on the tape, since the dump is written to file number zero.

Diskette Dump

To dump to diskette, follow these steps:

1. For a single-density diskette on the primary controller, select unit number three on the diskette drive. For a single-density diskette on the secondary controller, select unit number three on the diskette drive. For a double-density diskette, select the unit number and controller specified to SYSGEN. Make sure that no other diskette drive has the same unit number.

2. Tape the write-protect hole of a Data General diskette (or other diskette that has been hardware formatted); insert this diskette in the drive. Shut the door and turn the diskette drive ON.

3. Press the CPU's CONTINUE switch. The dump program then displays the message READY?

4. Again press the CPU switch CONTINUE. The dump routine copies memory to the diskette; if it displays the messages DONE and READY, proceed to step nine.

5. If all addresses cannot fit on one diskette, the program displays the message REPLACE, followed by READY? Open the diskette door, remove the diskette, insert another hardware-formatted diskette in the drive, and close the door. Press the CPU switch CONTINUE. The program then copies the rest of memory to the second diskette, and displays the messages DONE and READY?

6. The diskette dump is complete. To stop the program, press the CPU switch STOP; to produce another dump, remove the diskette, and repeat steps three, four, and five.

7. If you have forgotten a step, the program displays the message ERROR, then READY? Execute the step and press CONTINUE.

CORE DUMPS taken from a machine with the maximum amount of memory supported by RDOS (2048 KB) will fit onto two double-density diskettes. If dumping to single-density diskette, eight diskettes will be needed. When the first diskette is full, the message REPLACE followed by UNIT X READY? will be displayed on the console. Replace the diskette and lift the CONTINUE switch. The diskette dump cannot be copied under RDOS. In order to read a diskette dump as an RDOS file, the diskette must have been fully initialized with the disk initialization program, DKINIT, before the dump was taken. Then, after taking the dump, bring up RDOS on another RDOS disk. Type INIT/F to the diskette containing the core dump. Next, create a three-block, contiguous file on this disk with the CCONT command. Create a second continguous file, without zeroing the data blocks, by issuing the CCONT command followed by the /N switch. The number of blocks in this file should be equal to four times the size of memory. This file will contain the core dump.

When the core dump is complete, the message DONE, followed by UNIT X READY? appears on the master console. To replicate the core dump, mount another tape or insert another diskette (as appropriate); ensure that the device is ready to receive a core dump, and press the CONTINUE key on the front panel. The core dump procedure will then be repeated.

If an error is encountered (ie, unit off line, etc.), the message ERROR! followed by UNIT X READY? appears on the system console, and the CPU halts. After the error has been corrected, press CONTINUE and the process will automatically restart.

Appendix F

Page Zero and Hardware Reserved Locations

This appendix provides a listing of the page zero and hardware reserved locations from the PARS file.

```
01                          ;
02                          ; COPYRIGHT (C) DATA GENERAL CORPORATION 1977, 1978, 1979, 1980, 1981, 1982, 1983
03                          ;
04                          ; ALL RIGHTS RESERVED.
05                          ; LICENSED MATERIAL-PROPERTY OF DATA GENERAL CORPORATION.
06                          ;
07                          ; = = = = = = = = = = = = = = = = = = = = = = = =
08                          ; RDOS REVISION 07.10 SYSTEM PARAMETERS
09                          ; = = = = = = = = = = = = = = = = = = = = = = = =
10
                                        .TITLE PARS
12
13                          .
 003 PARS
01                          ;
02                          ; PAGE ZERO
03                          ;
04
05              000000      .DO ?ANSW
06
07                          ;.DUSR      SYST=     2          ; SYSTEM CALL ADDRESS
08                          ;.DUSR      NSTOV=    3          ; NOVA STACK OVER FLOW VECTOR
09                          ;.DUSR      .RTN=     4          ; ADDRESS OF RETURN ROUTINE
10                          .DUSR       CC=       5          ; CURRENT CELL
11                          .DUSR       RLOC=     6          ; PAGE ZERO TEMP.
12                          .DUSR       .SAV=     7          ; ADDRESS OF SAVE ROUTINE
13                          .DUSR       CSP=      10         ; STACK POINTER
14                          .DUSR       .PNIC=    11         ; PANIC
15                          ;.DUSR      USTP=     12         ; USTP DEFINED IN PARU
16                          .DUSR       CQ=       13         ; CURRENT TASK QUEUE
17                          .DUSR       CRSEG=    14         ; PTR TO OVERLAY TABLE ENTRY
18                          .DUSR       CMSK=     15         ; CURRENT MASK
19
20                          .DUSR       HRBEG=    40         ; START OF HARDWARE RESERVED AREA
21                          ; DEFINED IN NSID.SR
22                          ;.DUSR      TRPC=     46         ;     INSTRUCTION TRAP PC FOR NOVA 3
23                          ;.DUSR      TRHN=     47         ;     INSTRUCCTION TRAP HANDLER FOR NOVA 3
24                          ;.DUSR      CSL=      42         ;     STACK LIMIT FOR NOVA 3
25                          ;.DUSR      CSO=      43         ;     STACK OVERFLOW HANDLER FOR NOVA 3
26
27                          .ENDC
28
29                          ;    29 OCTOBER 1980    PR5    SAF # R-101
30                          ; AUTO-INCREMENT TEST LOCATION FOR DETECTING ALPHA
31                          ; (MICRO-ECLIPSE)
32              000020      .DUSR       AITST=    20
33
34
35
```

Figure F.1 Listing of PARS, giving page zero and hardware reserved locations

```
36          000001      .DO ?ABSW
37
38                      ; HARDWARE RESERVED LOCATIONS
39          000040      .DUSR       HRBEG=    40          ; START OF HARDWARE RESERVED AREA
40                      ; DEFINED IN NEID.SR
41                      ;.DUSR      SP=       40          ; STACK POINTER
42                      ;.DUSR      CSP=      41          ; FRAME POINTER (LOGICAL STACK PTR)
43                      ;.DUSR      CSL=      42          ; STACK LIMIT
44                      ;.DUSR      CS0=      43          ; STACK OVERFLOW ROUTINE PTR
45                      ;.DUSR      XOPA=     44          ; XOP ORIGIN ADDRESS
46                      ;.DUSR      FPFA=     45          ; FLOATING POINT FAULT ADDRESS
47                      ;
48                      ;; DEFINE OFFSETS FOR PAGE 0 INT STK LOCATIONS
49                      ;.DUSR      ISP=      4           ; SKP SP
50                      ;.DUSR      CMSK=     5           ; CURRENT MASK
51                      ;.DUSR      ISL=      6           ; LIMIT
52                      ;.DUSR      ISO=      7           ; OVERFLOR ADDR
53
54                      ; OTHER PAGE ZERO LOCATIONS
55                      ;           LOCATION    12 IS USTP (DEFINED IN PARU)
56                      ;           LOCATION     2 IS THE SYSTEM ENTRY POINT
57                      ;           LOCATION     3 IS THE PROTECTION FAULT ROUTINE POINTER
58          000010      .DUSR       CC=       10          ; CURRENT CELL
59          000011      .DUSR       .PNIC=    11          ; PANIC
60          000013      .DUSR       CQ=       13          ; CURRENT TASK QUEUE
```

Figure F.1 Listing of PARS, giving page zero and hardware reserved locations

Appendix G

Hollerith—ASCII Conversion Table

Char.	Card Code	ASCII Code
NUL	12-0-9-8-1	000
SOH	12-9-1	001
STX	12-9-2	002
ETX	12-9-3	003
EOT	9-7	004
ENQ	0-9-8-5	005
ACK	0-9-8-6	006
BEL	0-9-8-7	007
BS	11-9-6	010
HT	12-9-5	011
LF	11-9-5	012
VT	12-9-8-3	013
FF	12-9-8-4	014
CR	12-9-8-5	015
SO	12-9-8-6	016
SI	12-9-8-7	017
DLE	12-11-9-8-1	020
DC1	11-9-1	021
DC2	11-9-2	022
DC3	11-9-3	023
DC4	4-8-9	024

Char.	Card Code	ASCII Code
NAK	9-8-5	025
SYN	9-2	026
ETB	0-9-6	027
CAN	11-9-8	030
EM	11-9-8-1	031
SUB	9-8-7	032
ESC	0-9-7	033
FS	11-9-8-4	034
GS	11-9-8-5	035
RS	11-9-8-6	036
US	11-9-8-7	037
SPACE	NO PUNCHES	040
!	11-8-2	041
"	8-7	042
#	8-3	043
$	11-8-3	044
%	0-8-4	045
&	12	046
'	8-5	047
(	12-8-5	050
)	11-8-5	051

Char.	Card Code	ASCII Code
*	11-8-4	052
+	12-8-6	053
,	0-8-3	054
-	11	055
.	12-8-3	056
/	0-1	057
0	0	060
1	1	061
2	2	062
3	3	063
4	4	064
5	5	065
6	6	066
7	7	067
8	8	070
9	9	071
:	8-2	072
;	11-8-6	073
⟨	12-8-4	074
=	8-6	075
⟩	0-8-6	076
?	0-8-7	077
@	8-4	100
A	12-1	101
B	12-2	102

Char.	Card Code	ASCII Code
C	12-3	103
D	12-4	104
E	12-5	105
F	12-6	106
G	12-7	107
H	12-8	110
I	12-9	111
J	11-1	112
K	11-2	113
L	11-3	114
M	11-4	115
N	11-5	116
O	11-6	117
P	11-7	120
Q	11-8	121
R	11-9	122
S	0-2	123
T	0-3	124
U	0-4	125
V	0-5	126
W	0-6	127
X	0-7	130
Y	0-8	131
Z	0-9	132
[	12-0-5-8	133
\	0-8-2	134

Char.	Card Code	ASCII Code
]	12-11-5-8	135
¬ or ↑	11-8-7	136
— or ←	0-8-5	137
\	8-1	140
a	12-0-1	141
b	12-0-2	142
c	12-0-3	143
d	12-0-4	144
e	12-0-5	145
f	12-0-6	146
g	12-0-7	147
h	12-0-8	150
i	12-0-9	151
j	12-11-1	152
k	12-11-2	153
l	12-11-3	154
m	12-11-4	155
n	12-11-5	156
o	12-11-6	157
p	12-11-7	160
q	12-11-8	161
r	12-11-9	162
s	11-0-2	163
t	11-0-3	164
u	11-0-4	165
v	11-0-5	166
w	11-0-6	167
x	11-0-7	170
y	11-0-8	171
z	11-0-9	172
{	12-0	173
	12-7-8	174
	11-0	175
~	11-0-1	176
DEL	12-9-7	177

Appendix H

ASCII Character Set

To use this chart in octal, find the character whose code you want, then read straight up the column. The figures at the top are the first two digits. Now, return to the character and read the figure at the far left of its row; this is the third digit in the octal code. (In the legend, the octal code for @ is 100).

DECIMAL	OCTAL	HEX	KEY SYMBOL	MNEMONIC
0	000	00	↑@	NUL
1	001	01	↑A	SOH
2	002	02	↑B	STX
3	003	03	↑C	ETX
4	004	04	↑D	EOT
5	005	05	↑E	ENQ
6	006	06	↑F	ACK
7	007	07	↑G	BEL
8	010	08	↑H	BS (BACKSPACE)
9	011	09	↑I	TAB
10	012	0A	↑J	NEW LINE
11	013	0B	↑K	VT (VERT TAB)
12	014	0C	↑L	FORM FEED
13	015	0D	↑M	CARRIAGE RETURN
14	016	0E	↑N	SO
15	017	0F	↑O	SI
16	020	10	↑P	DLE
17	021	11	↑Q	DC1
18	022	12	↑R	DC2
19	023	13	↑S	DC3
20	024	14	↑T	DC4
21	025	15	↑U	NAK
22	026	16	↑V	SYN
23	027	17	↑W	ETB
24	030	18	↑X	CAN
25	031	19	↑Y	EM
26	032	1A	↑Z	SUB
27	033	1B	ESC	ESCAPE
28	034	1C	↑\	FS
29	035	1D	↑]	GS
30	036	1E	↑↑	RS
31	037	1F	↑—	US

DECIMAL	OCTAL	HEX	KEY SYMBOL
32	040	20	SPACE
33	041	21	!
34	042	22	" (QUOTE)
35	043	23	#
36	044	24	$
37	045	25	%
38	046	26	&
39	047	27	' (APOS)
40	050	28	(
41	051	29	)
42	052	2A	*
43	053	2B	+
44	054	2C	, (COMMA)
45	055	2D	-
46	056	2E	. (PERIOD)
47	057	2F	/
48	060	30	0
49	061	31	1
50	062	32	2
51	063	33	3
52	064	34	4
53	065	35	5
54	066	36	6
55	067	37	7
56	070	38	8
57	071	39	9
58	072	3A	:
59	073	3B	;
60	074	3C	<
61	075	3D	=
62	076	3E	>
63	077	3F	?
64	100	40	@

DECIMAL	OCTAL	HEX	KEY SYMBOL
65	101	41	A
66	102	42	B
67	103	43	C
68	104	44	D
69	105	45	E
70	106	46	F
71	107	47	G
72	110	48	H
73	111	49	I
74	112	4A	J
75	113	4B	K
76	114	4C	L
77	115	4D	M
78	116	4E	N
79	117	4F	O
80	120	50	P
81	121	51	Q
82	122	52	R
83	123	53	S
84	124	54	T
85	125	55	U
86	126	56	V
87	127	57	W
88	130	58	X
89	131	59	Y
90	132	5A	Z
91	133	5B	[
92	134	5C	\
93	135	5D	]
94	136	5E	↑ OR ^
95	137	5F	← OR _
96	140	60	` (GRAVE)

DECIMAL	OCTAL	HEX	KEY SYMBOL
97	141	61	a
98	142	62	b
99	143	63	c
100	144	64	d
101	145	65	e
102	146	66	f
103	147	67	g
104	150	68	h
105	151	69	i
106	152	6A	j
107	153	6B	k
108	154	6C	l
109	155	6D	m
110	156	6E	n
111	157	6F	o
112	160	70	p
113	161	71	q
114	162	72	r
115	163	73	s
116	164	74	t
117	165	75	u
118	166	76	v
119	167	77	w
120	170	78	x
121	171	79	y
122	172	7A	z
123	173	7B	{
124	174	7C	|
125	175	7D	}
126	176	7E	~ (TILDE)
127	177	7F	DEL (RUBOUT)

Figure H.1 ASCII character set

DG-05495

Appendix I

Advanced Multitask Programming

For most multitask application programs, the features described in Chapter 5 of this manual are sufficient. This appendix is intended for users who want to write their own multitasking primitives (task calls), or whose tasks require one or more special resources, such as floating-point hardware, that the system does not provide for in a TCB. All discussions assume a familiarity with the material covered in Chapter 5.

The features described in this appendix can (1) provide more programming flexibility than the standard features alone, without requiring you to modify the task monitor sources; and (2) provide this flexibility in a system-independent way. You can use the calls in this appendix to develop application programs for any system configuration—RDOS or RTOS, mapped or unmapped. All you need do to reconfigure for a different system is load a program, via RLDR, with the appropriate system libraries.

Definitions

The following definitions relate to tasks and task states; they apply throughout RDOS and RTOS.

General Terms

Task Resources are those storage elements of the computer, such as accumulators and special memory locations, that two or more tasks must share. The task scheduler allows such sharing by ensuring that the proper values for each task's resources appear in the actual storage elements of the computer while the task is executing. When a task is not executing, the current values of its resources are held in its TCB.

Rescheduling is the process of selecting and executing the ready task of highest priority. The task scheduler performs rescheduling after each task call, after receiving control from the system following an interrupt, and when a system call completes. You can suppress rescheduling via task calls .DRSCH or .SINGLE, or by entering the scheduler state, as described later in this section. If you have not disabled rescheduling, you must assume that it can happen at any time.

Task swaps occur during rescheduling, when the task scheduler determines that it should execute a different task from the last one executed.If the last task to execute was not terminated, the scheduler saves the current state of its resources in its TCB. Then the scheduler restores to a new task its resources, in their former state, from *its* TCB. The scheduler places the new task's TCB in the active TCB chain at the end of its priority class; thus, the next time rescheduling occurs, this task will be considered for execution only after all others in its class. Finally, the new task receives CPU control and becomes the current task.

CTCB is a location maintained by the scheduler containing the address of the current task's TCB. If no task is currently active—for example, if all tasks are suspended or rescheduling is occuring—CTCB contains the address of the most recent task's TCB, as long as that task was not terminated; otherwise CTCB contains 0. Thus, CTCB identifies the task to which the current values of task resource storage elements belong; zero means that these values are no longer relevant.

CTCB is a page zero location. You can access it as folows to obtain the TCB address for the current task:

```
.EXTD CTCB
LDA ac, CTCB
```

Location USTCT in the user status table (UST) also contains the current TCB address. However, you should use CTCB instead of USTCT.

Hardware stacks are the storage elements of the computer with built-in stack functions. The hardware stack on an ECLIPSE computer occupies locations 40_8 through 43_8. On a NOVA 3 or microNOVA computer, the stack occupies the stack and frame pointers and location 42_8, which the system interprets as the stack limit. RDOS treats the hardware stack as a task resource, thereby making it available for use by all tasks.

A reentrant section of code (sequence of instructions) allows another task to enter this code before the original task exits. Code which several tasks can access is reentrant only if each task has its own local storage, which no other task executing the code can access. Reentrancy is commonly achieved by giving each task its own stack area and using the stack for local storage.

State Definitions

User state is the normal state for an application program. This is the state from which system and task calls are made, as described in Chapter 5. Code must be reentrant in user state if more than one task will use it. In this state, task execution is suspended on an interrupt if a higher-priority task is ready for execution. A task in user state can use the user stack pointer (USP) and the hardware stack; it can also examine, but not modify, CTCB and the current TCB. In dual-ground operation, it can determine the current ground by examining USTPC in the UST; USTPC contains 0 for the background, or 1 for the foreground. If there are no indicators of other states, the program is in user state.

Single-task state is used occasionally for a critical section of an application program. You enter this state via task call .SINGL; it prevents other tasks from gaining control. However, interrupts and the other ground (if any) continue to execute. A task can issue system calls from single-task state as well as from user state; it can also issue any task call except .MULTI, kill, or suspension commands. If a task issues .MULTI, or kills or suspends itself, the progrm enters user state. Code executed from single-task state need not be reentrant. It can use USP, the hardware stack, CTCB, and the current TCB as it can in user state. If location SM.SW contains a nonzero value, the program is in single-task state.

Scheduler state is the normal state for task call code. An interrupt can cause temporary loss of control, but, unlike user and single-task states, control returns to the point of interruption without rescheduling. Thus, scheduler state ensures that no other task in the same ground will get control, although interrupts and the other ground continue. Code executed in scheduler state need not be reentrant. It should not use USP or the hardware stack; but it can both read and modify CTCB and the current TCB, subject to restrictions described later. In unmapped RDOS systems, code cannot use USTPC to distinguish foreground from background; instead, it should compare the UST base (USTAD) to the value 400_8. A value of 400_8 for USTAD indicates the background; a value other than 400_8 indicates the foreground. A task is in scheduler state for unmapped RDOS if location USTPC contains a value other than 0 or 1; or, for mapped RDOS, if location 1 is nonzero. For RTOS, location .SYS is nonzero in scheduler state.

Interrupt-disable state is used to perform critical manipulation of TCB data or the active TCB chain. There is no way for a task in this state to lose control of the CPU, even temporarily.

Coding Your Own Task Calls

This section describes components of the task control block available for your use, along with the following, task scheduler commands:

EN.SCHED	Enter the task scheduler state.
.TSAVE	Save the task state.
RE.SCHED	Take normal exit from task scheduler state.
ER.SCHED	Take abnormal exit from scheduler state.
INT.DS	Enter interrupt-disabled state.
INT.EN	Exit from interrupt-disabled state.
ID.SRCH	Search for a task of a given priority.

TCB and Status Bits

Two status bits of word TPRST in a TCB are allocated for your use; you can use them to extend the standard features. Bit TSUPN, the user suspend bit, prevents a task from running when set. Bit TSUSR, the user status bit, does not affect task readiness but is available for storing an additional piece of task-related information.

Word TELN is also available for your own use. This word is typically employed to store the address of a TCB extension, allowing you to store as much additional, task-related information as you need.

Scheduler Calls

Like task calls, the scheduler commands defined here are external symbols that must be identified as such via an .EXTN statement in your source program. The relocatable loader, RLDR, resolves them at load time, according to system type. Each version of SYS.LB defines the scheduler commands for its version of the system—RDOS or RTOS, mapped or unmapped, and NOVA or ECLIPSE computers.

EN.SCHED

Enter scheduler state

Use the following format to enter scheduler state from user or single-task state:

```
;AC3 NOT EQUAL TO 0 AND NOT EQUAL TO 1
;FOR UNMAPPED RDOS SYSTEMS.
EN.SCHED
;RETURNS HERE WITH ALL ACS AND CARRY PRESERVED.
```

A task already in scheduler state can safely reissue EN.SCHED, but no change in state will occur.

.TSAVE

Task state save

For a task in scheduler state, this command saves the ACs, carry, and program counter in its TCB. The PC saved is the value in bits 1 through 15 of AC3 at the time of the last EN.SCHED command. The format for .TSAVE is as follows:

```
;ACS, CARRY, PC TO BE SAVED.
.TSAVE
;RETURNS HERE WITH AC0, AC1, AND CARRY UNCHANGED,
; AND AC2 = VALUE THAT WAS IN AC3
; AT TIME OF LAST EN.SCHED;
; AT AC3 = TCB ADDRESS
```

The .EN.SCHED and .TSAVE commands are meant to be used together at the start of code which implements a user-designed task call. For a task call with error return, they might be used as follows:

```
          .ENT .TASK, T.ASK
          .EXTN EN.SCHED, .TSAVE
          .ZREL
.TASK=    JSR@
          T.ASK
          .NREL
T.ASK:    INC 3,3          ;ASSUME NORMAL RETURN.
          EN.SCHED         ;ENTER SCHEDULER STATE.
          TSAVE            ;SAVE TASK STATE.
```

For a task call without an error return, this code would omit the increment instruction, INC.

RE.SCHED

Leave scheduler state normally

When you successfully complete the processing for a task call, issue the RE.SCHED command to exit to the scheduler for rescheduling. Use RE.SCHED in scheduler state, according to the following format:

```
;NO INPUT.
RE.SCHED
;NO RETURN.
```

ER.SCHED

Leave scheduler state abnormally

When you detect an error during task call processing, place an error code in AC2 and exit to the scheduler via the ER.SCHED command. This returns control to the location preceding the one specified by TPC, and passes back the error code in AC2. Use ER.SCHED in scheduler state, according to the following format:

```
;AC2 = ERROR CODE.
ER.SCHED
;NO RETURN.
```

INT.DS

Enter interrupt-disabled state

Use this command to enter interrupt-disabled state from scheduler state. Its format is as follows:

```
;NO INPUT
.INT.DS
;RETURNS HERE WITH AC0, AC1, AC2,
;AND CARRY UNCHANGED.
```

INT.EN

Leave interrupt-disabled state

Use this command to leave interrupt-disabled state and return to scheduler state. Its format is as follows:

```
;NO INPUT.
INT.EN
;RETURNS HERE WITH AC0, AC1, AC2,
;AND CARRY UNCHANGED.
```

ID.SRCH

Task ID search

Use this command to search for a task with a given ID. You can issue ID.SRCH in either scheduler or interrupt-disabled state. Its format is as follows:

```
;RIGHT BYTE OF AC1 = ID OF SOUGHT TASK.
ID.SRCH
;ERROR RETURN HERE, WHERE AC2 = ERROR CODE.
;NORMAL RETURN HERE, WHERE
;AC2 = TCB ADDRESS OF SOUGHT TASK.
```

For both returns, AC0 and carry are preserved, while the left byte of AC1 is zeroed and the right is preserved.

Handling Additional Task Resources

This section explains how to manage task resources that are not automatically managed by the system. It contains three discussions. At certain points in its scheduling process, the scheduler calls out to routines which you may supply to handle your additional task resources. These callouts are described first.

Second, if floating-point hardware and/or a block of contiguous memory locations are among the resources you need, you can simply use a handler supplied in SYS.LB, as discussed under "Additional Resource Handler."

Third, under "Operator Communications," the method of handling additional task resources while using the operator communications package (OPCOM) is explained.

Task Scheduler Call-Outs

To use any callout described here, write, assemble, and load a routine of the appropriate name and function. You must insert the name of the routine in the RDLR command line before the loader program searches SYS.LB. (By default, this search occurs at the end of the command line.) If you do not supply a routine, RDLR loads a dummy routine, which does nothing, from SYS.LB.

TSK.X

Task initiation callout

This callout allows you to endow a new task with additional resources. When the scheduler initates a task, it first removes a TCB from the free TCB chain. Then it initializes certain parts of the TCB, as described in a later section titled "Task Control Block Values." The scheduler then calls out to your TSK.X routine in scheduler state.

Your TSK.X routine can initialize certain parts of the TCB and change other parts initialized by the scheduler (subject to restrictions noted under "Task Control Block Values"). On a normal return, the scheduler links the TCB for the new task, as modified by your TSK.X code, into the active TCB chain.

The scheduler transfers control to address TSK.X with the accumulators set up as follows:

AC0 Contains the value passed to .TASK in AC2. AC0 is irrelevant if .QTSK initiates the task.

AC1 Contains -1 if .TASK initiates the task, or the address of the task queue table if .QTSK initiates the task.

AC2 Contains the address of the new task's TCB.

AC3 Contains the (error) return address.

The routine you supply with entry address TSK.X need not preserve accumulators or carry. If you detect an error, place an error code in AC2 and return control to the location whose address you received in AC3. On a normal return, pass control to the location whose address is one greater than the one you received in AC3, for example:

```
        .ENT TSK.X

        .NREL
TSK.X:  STA 3, RTNAD        ;SAVE RETURN.
        .
        .
        COM # 1,1SZR        ;.TASK OR .QTSK?
        JMP QUE

        .
        .
QUE:    .                   ;HANDLE .QTSK CASE.
        .
.BAD:   LDA 2, CODE         ;ERROR RETURN.
        JMP @RTNAD

GOOD:   ISZ RTNAD           ;NORMAL RETURN.
        JMP @RTNAD

RTNAD:  .BLK 1
```

When you return an error indication and .TASK is the initiator, the task is not initated, and its TCB returns to the free TCB chain; the error code that you place in AC2 is passed to the task which issued .TASK. When you return an error indication and .QTSK is initiating the task, the system tries again one second later.

TRL.X

Task termination callout

This callout frees a task's additional resources when it terminates—typically those resources that you assigned in a TSK.X routine. The scheduler calls this routine in scheduler state whenever a task is being killed, with the task's TCB already unlinked from the active chain but not yet restored to the free chain. The scheduler transfers control to address TRL.X, with ACs set up as follows:

AC2 Contains the TCB address of the task being killed.

AC3 Contains the return address.

The routine supplied with entry address TRL.X need not preserve accumulators or carry. When you have finished your processing, return control to the location whose address you received in AC3. There is no way to signal an error from TRL.X.

ESV.X

Task swap callout

This callout allows you to save and restore additional task resources as needed when a task swap occurs. The scheduler calls the routine in scheduler state and transfers control to address ESV.X, with the accumulators set up as follows:

AC2 TCB address for the task losing control, or 0 if no task is losing control (as described in the definition of CTCB, earlier)

.CTCB TCB address of the task gaining control.

AC3 Return address.

The routine supplied with entry address ESV.X need not preserve accumulators or carry. When you have finished processing, return control to the location whose address you received in AC3. There is no way to signal an error from ESV.X.

A 0 passed to you in AC2 indicates that no valid, most recently active task exists whose resources should be saved. This situation occurs as the default task is initially selected for execution. ESV.X will be called with 0 in AC2, and the TCB address for the default task in CTCB. It also occurs after a task terminates, because its resources were freed by TRL.X and are no longer meaningful to ESV.X.

Additional Resource Handler

The system library, SYS.LB, contains an ESV.X routine that provides, in part, for the additional task resources of floating-point hardware and a block of contiguous storage words. To load this module, insert the statement .EXTN ESV.X in any source module whose name will occur in the RLDR command line before SYS.LB is searched.

For each task requiring access to the floating-point hardware, you must provide a block of words to store the task's values for its floating-point state. The size and content of this block depend on the kind of computer you use. For an ECLIPSE computer, the block has this format:

Status	2 words
FPAC0	4 words
FPAC1	4 words
FPAC2	4 words
FPAC3	4 words

This format matches the one used by the FPSH and FPOP instructions. For a NOVA computer, storage block has the following format:

FPAC	4 words
TEMP	4 words
Status	1 word

To provide for a block of contiguous, memory locations as an additional task resource, you must define two symbols via .ENT and give them the following values:

ESV.S	Equals the starting address of the block.
ESV.Z	Equals the number of words in the block.

In addition, you will need to provide a block of memory, whose length in words equals the value of ESV.Z, for each task that is to use this additional resource.

Finally, for each task that will use either the floating-point hardware or contiguous memory locations, you must initialize offset TELN in the TCB, within the TSK.X routine that you must supply. The value placed in TELN depends on the task's needs.

- If TELN contains either 0 or 10000_8, neither the floating-point nor the contiguous memory resources will be handled. RDOS initializes TELN to 0; thus, you need not change it for a task requiring neither resource.
- If the task requires floating-point hardware but not contiguous memory, set TELN to the indirect address of the appropriate floating-point block described earlier.
- If the task needs contiguous memory but not floating-point hardware, set TELN to the (direct) address of a block of words ESV.Z + 1 words long, and set the first of these words to either 0 or 100000_8. The contiguous memory locations will be saved in the remaining words of this block.
- If the task requires both resources, set TELN as described immediately above, but set the first word of the block to the (direct) address of a floating-point save area.

When initializing TELN, you can also initialize the contents of these save areas. The values that your TSK.X routine places in these areas will be the initial values when the task being initiated starts executing.

Restrictions and Warnings

The system-supplied, additional resource handler assumes that TELN is set up properly for either or both of the resources; it does not prevent an unprepared task from using one of these resources inadvertantly. If this occurs, results are unpredictable.

For a task to use these resources, you must set up the task's TELN in your TSK.X routine. You cannot change a task's TELN after the task has been initiated.

Extra Resources

If a task requires resources in addition to floating-point hardware and contiguous memory locations, you can write your own ESV.X routine to handle the extra resources, using the system-supplied handler as a subroutine. From within your own ESV.X routine, call out to the supplied handler using the alias ESV.A instead of ESV.X, with the accumulators set up appropriately.

Operator Communications

When issuing a task call .QTSK, you must pass in AC2 the address of a task queue table, whose format and length conform to the description in Chapter 5. When the scheduler calls out to TSK.X, it passes the queue table address in AC1. Thus, you can append additional information to the queue table—that is, supply a longer table—and access this information from within TSK.X.

The operator communications feature (OPCOM) describes programs to be run from the console by means of a program table consisting of program frames of a given length. When the operator types a QUE command, the scheduler copies information from a program frame into a queue table. You can increase the size of a program frame, causing the scheduler to pass additional information about a program to TSK.X via a longer queue table. To do this, define symbol LPN.X via .ENT, assigning it a value equal to the number of additional words in each program frame. On an OPCOM QUE command, these words will be copied, in order, to the end of the associated task queue table, where they will be accessible to TSK.X.

Task Control Block Values

Table I.1 describes the initial values that the scheduler assigns to words in a TCB, and when these values can be changed during a task's lifetime. Each name is a symbol in PARU.SR representing the offset within the TCB. Initial contents are values placed in a TCB word by the task scheduler and seen on input to TSK.X. Each value applies to both .TASK and .QTSK unless two of them are separated by a slash (/); in this case, the first entry applies to .TASK and the second to .QTSK. A "Yes" under column .TASK? means that TSK.X can set or change the contents of this word if .TASK is initiating the task; "No" means that TSK.X can not change this word. A "Yes" or "No" in column .QTSK? means the same thing as applied to .QTSK. A "Yes" in column "Later?" means that this word can be changed later in the task's life; "No" means it cannot be changed. Each italicized number refers to a note at the end of the table.

Name	Initial Contents	.TASK?	.QTSK?	Later?
TPC	B0-14: Start addr; B15: Undefined	Yes	Yes	Yes
TAC0	Undefined/System-maintained	Yes	No	Yes
TAC1	Undefined/System-maintained	Yes	No	Yes
TAC2	AC2 at .TASK/System-maintained	Yes	No	Yes
TAC3	K.ILL[1] /System-maintained[2]	Yes	No	Yes
TPRST	B0-7: 0; B8-15: Start pri.	Yes	Yes	*3, 4*
TSYS	System-maintained	No	No	No
TLNK	System-maintained	No	No	No
TUSP	Undefined	Yes	Yes	*5*
TELN	0	Yes	Yes	*6*
TID	Task identifier	No	No	No
TTMP	System-maintained	No	No	No
TKLAD	0	Yes	Yes	Yes
TSP	Undefined	Yes	Yes	*5*
TFP	Undefined	Yes	Yes	*5*
TSL	Undefined	Yes	Yes	*5*
TSO	Undefined	Yes	Yes	*5*

Table I.1 TCB words and how they can be changed

[1]Address K.ILL is the entry for the .KILL task call code. This address is placed in TAC3 so that a task can kill itself by simply returning to the address it receives in AC3.

[2]At TSK.X time for a task initiated by Q.TSK, TAC3 does not contain the address K.ILL. However, after TSK.X completes but before the new task gains control, the scheduler places the address K.ILL in TAC3, so that the task's initial AC3 will be correct. (See also note 1.)

[3]The interrupt world can modify the status bits on suspended tasks only. Thus, modifying status bits of a ready task must be a "task-indivisible" operation, while modification of a suspended task must be an "interrupt-indivisible" operation. The scheduler and interrupt-disabled states provide task-indivisibility. Interrupt-disabled state provides interrupt-indivisibility, as does the use of bit instructions on an ECLIPSE computer.

[4]Do not modify the priority portion of TPRST; use task call .PRI instead.

[5]Because these values are saved and restored only on task swaps, it is meaningless to change them while in scheduler state. Instead, you should change the actual storage locations. Change USP (16_8) instead of TUSP. On an ECLIPSE computer, change locations 40 through 43 octal instead of TSP through TSO. On a NOVA computer with hardware stack, change the hardware stack and frame pointers, and locations 42_8 (stack limit) and 46_8 (instruction trap PC).

[6]As mentioned earlier, you cannot change word TELN after TSK.X time if you use the additional resource handler (ESV.X routine) supplied in SYS.LB.

Index

A

A (attribute protect) 9
.ABORT (abort a task) 126
Accumulators, status upon return from system calls 34
Active chain 120
Additional resource handler 249
Addressable memory, see logical address space
Addresses, mapped and unmapped system 3
.AKILL (kill all tasks of a given priority) 126
Aliases, link entries 19
ALMSPD.SR (line characteristics sourcefile) 30
.APPEND (open file for appending) 60
.ARDY (Ready all tasks of a given priority) 128
ASCII character set 239
Assembler cross-reference listing, see PARU.SR
Assembly language programs, examples 215
Assembly language source filename extensions 9
Assembly language, executable program files 2
.ASUSP (suspend all tasks of a given priority) 128
Asynchronous communications multiplexor (QTY), see multiplexors
Asynchronous line multiplexor (ALM), see multiplexors

B

Background memory, introduction to 2
Background program, checkpoints in 161
Background programming, introduction to 2
Bad block pool, location on disk 13
Binary file, definition of 7
Bits and associated device characteristics (Table 3.7) 53
Block access, contiguous file 13
.BOOT (bootstrap a new operating system) 179
BOOT.SV 13
Bootstrap root, location on disk 13
BOOTSYS.OL 17
.BREAK (interrupt program and save main memory) 86

C

C (contiguously organized file) 10
.CCONT (create contiguously organized file) 45
.CDIR (create a subdirectory) 42
Chained programs, definition of 91
Chaining process (Figure) 94
Chaining, introduction to 2
Channel selection 34
  example 35
Channels, multiple 29
.CHATR (change a file's attributes) 51
Checkpoint procedure, example 170
.CHLAT (change link access entry attributes) 55
.CHSTS (get the file directory information for a channel) 50
CLI commands
  CDIR 15
  CHATR 9
  CHATR 20
  CHLAT 20
  CRAND 10
  CREATE 10
  DEB 2
  DELETE 20
  DIR 17
  DUMP 26
  EQUIV 17
  EXFG 159
  INIT 17
  INIT/F 25
  LINK 18
  LIST/A 10
  LOAD 26
  on MCA lines 180
  POP 91
  RELEASE 17
  tuning 189
  UNLINK 20
  XFER 10
CLI levels 92
CLI LINK command examples 18
CLI, function of 1
CLI, use to organize user disk space 15
.CLOSE (close a file) 61
Closing down the operating system, see system shutdown
Commands
  clock 138
  commonly used 35
  file maintenance 45

.CONN (create contiguously organized file) 46
Console I/O commands 72
Contiguous file 10
Contiguous file block organization (Figure) 13
Contiguous file block organization, block access speed 16
Contiguous file block organization, definition of 13
Contiguous memory locations block, to provide 249
Control calls (Table 3.2) 37
Control characters interrupts, see keyboard interrupts
Controller support, disk drives 8
Core dump
  procedures for 229
  to produce 228
.CPART (create a secondary partition) 43
.CRAND (create randomly organized file) 46
.CREAT (create sequentially organized file) 47
Current directory, definition of 17

D

D (randomly organized file) 10
Data block structure (Figure) 24
Data channel, mapped system 3
Data encoding
  nine-track units (Figure) 24
  seven-track units (Figure) 24
.DDIS (disable user access of a device) 77
.DEBL (enable user access of a device) 76
.DELAY (delay execution of the calling task) 139
.DELET (delete a file) 47
DEQ (dequeue a previously queued task) 148
Device access 3
Device access commands 76
Device and directory commands 39
Device name
  in command lines 7
  reserved 7
.DIR (initialize a directory or device) 40
Direct block input/output 10
Direct block input/output transfers 13
Direct block mode 57
Directory commands (Table 2.3) 23
Disk block 10
Disk file characteristics (Table 3.6) 52
Disk file
  access 7
  attributes 9
  characteristics 10
  methods for finding 18
  to open 7
  to reference 18
Disk filename, definition of 9
Disk space
  apportionment (Figure) 16
  multiuser system 16
Diskette dump 231
DKINIT.SV 13
.DQTSK (dequeue a memory-resident or overlay task) 138
.DRSCH (disable rescheduling) 156
DSR (data set ready) signal 30
Dual processor, program communication 178
Dual programming 159
Dual programs 159
  in unmapped system (Figure) 163
  to execute 162
.DUCLK (define a user clock) 139

E

EN.SCHED (enter scheduler state) 243
End-of-file (EOF) marker 24
End-of-tape (EOT) marker 24
.EOPEN (open file for exclusive write access) 59
.EQIV (assign a temporary name to disk or tape unit) 43
ER.SCHED (leave scheduler state abnormally) 244
.ERDB (extended read direct block) 113
Error codes, exceptional status 227
Error summary 210
Errors from control calls (Table 3.3) 38
.ERSCH (reenable rescheduling) 156
.ERTN (return from program swap with call program's error status) 96
ESV.X (task swap) 248
.EWRB (extended direct write block) 114
EXAMPLE program listing (Figure) 219
Examples, line printer dump (Figure E.1) 230
.EXBG (checkpoint a mapped background program) 168
Exceptional status 227
.EXEC (swap or chain a save file into execution) 95
.EXFG (execute a program in the foreground) 164
EXFG command (execute a program in foreground) 03
Extended address space, window mapping and virtual overlays 91
Extended direct block I/O
  example 115
  overview of 112
Extended memory, see extended address space

F

.FGND (see if foreground program running and check level) 165
File access attributes 20
File access, change attributes 9
File attribute commands 51
File types, examples 7
File
  backup on magnetic tape 22
  index 12
  overview of 7
  random organization 12
  save 12
  transfer 10

Filename extensions 9
Filename.OL 97
Filename.SV 97
Floating point hardware, to manage task 249
Foreground memory, introduction to 2
Foreground program priority 2
Foreground program, to execute 162
Foreground programming, introduction to 2
Foreground/background system calls 164
Frame, definition of 14
Free element chain 120
Free form input/output modes 23
Free form mode 57
Full initialization, function of 17

G

.GCHAR (get a character) 72
.GCHN (get the number of a free channel) 61
.GCIN (get the input console name) 73
.GCOUT (get the output console name) 74
.GDAY (get today's date) 79
.GDIR (get current directory name) 42
.GHRZ (examine the system real time clock) 141
Global directory specifier 17
.GMCA (get current CPU's MCA number) 181
.GPOS (get the current file pointer) 62
.GSYS (get current operating system name) 44
.GTATR (get the file attributes and characteristics) 52
.GTOD (get the time of day) 78

H

High level languages, executable program files 2
Hollerith-ASCII conversion table 235

I

.ICMN (define a program communication area) 166
ID.SRCH (task ID search) 246
.IDEF (identify a user interrupt device) 172
.IDST (get a task's status) 141
Index blocks 12
.INIT (initialize a directory or device) 39
Initial disk block assignments 14
Initial disk block assignments (Table 2.2) 14
Input files, default 7
Input/output
  calls 56
  device names 7
  modes 23
INT.DS (enter interrupt-disabled state) 245
INT.EN (leave interrupt-disabled state) 245
.INTAD (reserve a program interrupt task) 88
  task program 84
Interprocessor buffer (IPB) 178
  introduction 177
  to program 178
Interrupt handler program (Figure) 83
Interrupt routines, to define 83
Interrupts, to service 171
.IOPC (initializing the operator communications
  package) 147
IPB, see interprocessor buffer
.IRMV (remove a nonSYSGENed interrupt device) 174
.IXMT (transmit a message from a user interrupt
  service) 130

K

Keyboard interrupts 82
KIL (kill a task) 149
.KILAD (define a kill-processing address) 125
.KILL (delete the calling task) 125

L

L (link entry) 10
.LEFD (disable the LEF mode) 193
.LEFE (enable the LEF mode) 194
.LEFS (get the LEF mode status) 194
LFE instruction, see mapped systems
Line characteristics, to define 31
Line mode 56
Line printer copy of file, to obtain 21
Line printer dump 229
Line sixty-four reads and writes 28
.LINK (create a link entry) 54
Link commands 53
Link entries
  aliases 18
  definition of 7
  overview of 19
Load effective address (LFE) instruction 193
Loading the overlay root programs (Figure) 99
Logical address
  mapped system 3
  sequentially organized files 11
  space 91
Lower case letters, RDOS conversion of 9

M

MAC.PS (macroassembler permanent symbol file) 6
Magnetic tape dump 231
Magnetic tape files 23
MAP unit, introduction of 3
.MAPDF (define a window and window map) 110
MAP.DR
  definition of 14
  function of 13
Mapped and unmapped memory (Figure) 05
Mapped system
  addresses 3
  affect of program swaps and chains 93
  extended block input/output 112

extending address space 103
load effective address (LFE) instruction 193
page length 3
protecting user memory 102
traps 3
window mapping 107
Master device, see master directory
Master directory 15
MCA, see multiprocessor communications adaptor
MCABOOT program 181
.MDIR (get name of master directory) 44
.MEM (determine available memory) 75
.MEMI (change NMAX) 75
Memory allocation commands 74
Memory allotment, see mapped system
Memory block size 102
Memory considerations 2
Memory extension with disk space 91
Memory location numbers and mnemonics 122
Minimum hardware to run RDOS 1
Modem support 29
Modes, input/output 56
.MTDIO (perform free format I/O) 70
.MTDIO values 71
.MTOPD (open a tape unit and file for free format I/O) 69
Muliplexors, to monitor line interrupts 28
.MULTI (restore the multitask environment) 155
Multi-task and single-task environments, definition of 33
Multiple processor line connections 180
Multiple processor systems 177
Multiplexor error messages (Table 2.6) 30
Multiplexor lines, condition for swapping 92
Multiplexors, bits that affect (Table 2.4) 28
Multiplexors, overview of 27
Multiprocessor communications adaptor (MCA)
introduction to 177
MCABOOT program 181
to program 180
Multiprocessor system, example 182
Multitask environment, disabling and enabling 154
Multitask programs, procedures for building 119
Multitask scheduler (TCBMON) 33

N

N (no links possible) 9
NMAX (highest address) 3
NMAX, affect upon program swaps 92
NREL (normal relocatable memory) 3

O

.ODIS (disable console interrupts) 87
OPCOM, see task
OPCOM command examples (Figure) 153
OPCOM commands
DEQ 148
KIL 149
PRI 149
QUE 150
RDY 151
RUN 151
SUS 152
TST 152
.OPEN (open a file) 57
.OPEN system call, function of 7
Operator communication module (OPCOM) 146
Output files, default 7
Overlay directory
place in memory 225
structure (Figure) 225
Overlays
directory of 97
file size 97
format to 97
in swaps and chains 91
management of 131
memory considerations 02
node size 97
procedures for using 97
program 91
user 97
user (Figure) 98
.OVEX (release an overlay and return to the caller) 134
.OVKIL (kill the calling task and release its overlay) 135
.OVLOD (load an overlay) 101
.OVOPN (open overlays for reading) 100
.OVREL (release an overlay) 134
.OVRP (replace overlays in an overlay file) 102

P

P (permanent file) 9
Page length, see mapped system
Page zero memory locations for user programs 3
Parameters, user 213
PARS, page zero and hardware reserved locations 233
Partial initialization, function of 17
Partitions and subdirectories 15
Partitions, primary and secondary 15
PARU.SR, assembler cross-reference listing 223
PARU.SR, file 213
.PCHAR (put a character) 73
Power fail-auto restart procedures 175
Power-up service, user devices 176
PRI (change a task's priority) 149
.PRI (change the calling task's priority) 127
Primary partition subdirectories 15
Program development, steps 2
Program frame, to increase size 250
Program return, affect on current program 92
Program swapping (Figure) 93
Program swaps and chains 91
Program swaps, definition of 91
Pseudo-op .ENTO 99
Push, see program swapping

Q

.QTSK (queue a memory-resident or overlay task) 135
.QTSK example (Figure) 137
QUE (queue a task for periodic execution) 150

R

R (read protect) 9
Random file 15
Random file block organization (Figure) 12
Random record mode 57
.RDB (read a series of disk file blocks) 68
.RDCMN (read a message from the other program) 167
.RDL (read a line) 63
.RDOPR (read an operator message) 168
RDOS command summary (Table A.1) 197
RDOS
  disk organization (Figure) 21
  executive 3
  features 1
  media 7
  minimum hardware 1
  organization 3
.RDR (read random record) 67
.RDS (read sequential) 65
.RDSW (read the front panel switches or register) 77
RDY (ready A task) 151
.REC (receive a message) 130
Relocatable binary file, see binary file
.REMAP (perform a logical window transfer) 111
.RENAM (rename a file) 48
RE.SCHED (leave scheduler state normally) 244
Reserved device names (Table 2.1) 08
.RESET (close all files) 62
Resolution file 18
RLDR
  function of 3
  in program load 2
.RLSE (release a directory or device) 41
Root program, definition of 97
.ROPEN (open file for reading only) 59
.RSTAT (get a file's current directory status) 57
.RTN (return to the program at the next higher level) 96
.RUCLK (remove a user clock) 140
RUN (execute a task) 151

S

S (save file) 9
Save file, definition of 7
Scheduler call, .TSAVE 243
.SDAY (set today's date) 80
Second controller, names 8
Secondary partitions, use of to prevent loss 16
Sequential file block organization 11
  (Figure) 11
Sequential file, definition 10
Sequential mode 56
.SINGL (disable the multitask environment) 154
SMEM (reserve memory in foreground) 3
.SMSK (modify the current interrupt mask) 174
.SPDA (disable device spooling) 81
.SPEA (enable device spooling) 82
.SPKL (stop a spool operation) 81
.SPOS (set the current file pointer) 63
.STAT (get a file's current directory status) 48
State, definitions 242
.STMAP (set the data channel map) 175
.STOD (set the time of day) 79
Subdirectories, see primary partition subdirectories
Subdirectories and partitions, maximum number allowed 17
Summaries, error code meanings 227
SUS (suspend a task) 152
.SUSP (suspend the calling task) 128
Swapping, introduction to 2
Symbolic debugger, use with file load 2
SYS.DR
  block composition 14
  contents 16
  function of 13
SYS.LB (system library)
  contents of 6
  memory considerations 2
SYSGEN, see system generation program
System
  buffer requirements 187
  cell requirements 186
  self-tuning 189
  stack requirements 186
  tuning 185
System and task calls
  definition of 33
  function of 1
  (Table 3.9) 88
  (Table 4.1) 116
System buffers
  location in memory 3
  use in file access 12
System calls
  .APPEND 60
  .BOOT 179
  .BREAK 86
  .CCONT 45
  .CDIR 15, 42
  .CHATR 51
  .CHLAT 20, 55
  .CHSTS 50
  clock/calendar 77
  .CLOSE 61
  commonly used (Table 3.1) 36
  .CONN 46
  .CPART 43
  .CRAND 10, 46
  .CREAT 47

.DDIS 77
.DEBL 76
.DELET 20, 47
.DIR 17, 40
.EOPEN 59
.EQIV 17, 43
.ERDB 113
.ERTN 96
.EWRB 114
.EXBG 168
.EXEC 95
.EXFG 164
.FGND 165
file attribute 51
file maintenance 45
.GCHAR 72
.GCHN 61
.GCIN 73
.GCOUT 74
.GDAY 79
.GDIR 42
generic format for 34
.GMCA 181
.GPOS 62
.GSYS 44
.GTATR 52
.GTOD 78
.ICMN 166
.IDEF 172
.INIT 17, 39
.INTAD 88
.IRMV 174
.LEFD 193
.LEFE 194
.LEFS 194
.LINK 18, 54
.MAPDF 110
.MDIR 44
.MEM 75
.MEMI 75
.MTDIO 24, 70
.MTOPD 24, 69
.ODIS 87
.OPEN 7, 57
.OVLOD 101
.OVOPN 100
.OVRP 102
.PCHAR 73
program swaps and chains 91
.RDB 68
.RDCMN 167
.RDL 63
.RDOPR 168
.RDR 67
.RDS 65
.RDSW 77
.REMAP 111
.RENAM 48
.RESET 62
.RLSE 17, 41
.ROPEN 59
.RSTAT 57
.RTN 96
.SDAY 80
.SMSK 174
.SPDA 81
.SPEA 82
.SPKL 81
.SPOS 63
.STAT 48
.STOD 79
.TUOFF 191
.TUON 191
.UIEX 173
.ULNK 20, 55
.UPDAT 50
.UPEX 173
.VMEM 110
.WRB 68
.WRCMN 166
.WREBL 104
.WRL 64
.WROPR 167
.WRPR 104
.WRR 68
.WRS 66
with multiplexors 27
System directory, see SYS.DR
System file names (Table 1.1) 6
System generation program (SYSGEN) 1
System library, see SYS.LB
System overlays and their functions (Table 9.1) 187
System shutdown
examples 22
procedures for 17

T

T (partition file) 10
Tape drive
initializing and releasing 24
rewinding with RELEASE command 26
Tape file
input/output modes 23
overwrite (Figure) 27
to link 27
to reference 26
write first file (Figure) 26
Task
clock commands 138
definitions 241
ID 117
initial 117
initiation 123
inter-task communication 129
managing by ID number 141
operator communication module (OPCOM) 146

overview 117
priorities 117
states 119
suspended 120
synchronization and communication 121
to delete 120
to enqueue 135
to lock a process 130
Task and system calls 123
.TASK (create a task) 124
Task calls
.ABORT 126
.AKILL 126
.ARDY 128
.ASUSP 128
.DELAY 139
.DQTSK 138
.DRSCH 156
.DUCLK 139
.ERSCH 156
.IDST 141
.IOPC 147
.IXMT 130
.KILAD 125
.KILL 125
.MULTI 155
operator communication 144
.OVEX 134
.OVKIL 135
.OVREL 134
.PRI 127
.REC 130
.RUCLK 140
.SINGL 154
single-task 33
.SUSP 128
.TASK 124
.TIDK 144
.TIDP 142
.TIDR 143
.TIDS 143
.TOVLD 132
to write 241
.TRDOP 145
.TWROP 145
.UCEX 140
.XMT and .XMTW 129
Task control block (TCB) 117
queues 120
structure (Table 5.1) 118
values 250
Task-processing modules, location in memory 4
Task resources, to handle additional 246
Task scheduler
enabling and disabling 155
location in memory 4
memory considerations 2
Task scheduler callouts, ESV.X 248
Task scheduler callouts, TRL.X 248
Task scheduler callouts, TSK.X 247
Task scheduler commands
ER.SCHED 244
ID.SRCH 246
INT.DS 245
INT.EN 245
RE.SCHED 244
summary of 242
Task state modification 127
TCB, see task control block
.TIDK (kill a task by ID number) 144
.TIDP (change a task's priority) 142
.TIDR (ready a task by ID number) 143
.TIDS (suspend a task by ID number) 143
TIMEC program listing (Figure) 217
.TOVLD (load a user overlay) 132
TOVLD logic sequence (Figure) 133
Traps
comparison with exceptional status reports 227
mapped system 3
.TRDOP (read a task message from the console) 145
TRL.X (task termination) 248
.TSAVE (task state save) 243
TSK.X (task initiation) 247
TST (display a task's status) 152
Tuning commands, CLI 189
.TUOFF (stop recording in the tuning file) 191
.TUON (start recording in the tuning file) 191
.TWROP (write a message to the console) 145

U

.UCEX (exit from a user clock routine) 140
UFD (user file descriptor) 14
template with displacement mnemonics (Table 3.4) 49
.UIEX (exit from a user interrupt routine) 173
ULM line codes 29
ULM line speed selection (Table 2.5) 29
.ULNK (get the file directory information for a channel) 55
Universal line multiplexor (ULM), see multiplexors
Unmapped RDOS, see unmapped system
Unmapped system
definition of 3
dual programs in 161
location of executive 3
.UPDAT (update a file) 50
.UPEX (exit from a power fail service routine) 173
User address space, to extend 2
User-defined attribute
& (ampersand) 9
? (question mark) 9
User directories 15
User file descriptors 14
User overlay management 131
User status table (UST), contents of 3
User status table (UST), structure of (Table 5.2) 121
User task queue table (Table 5.3) 136
UST, see user status table

V

Virtual overlays
  procedures for loading and remapping 102
  use to store subroutines 91
  with .OVLD 106
.VMEM (determine the number of free blocks) 110

W

W (write protect) 9
Window map
  to define and perform 107
  use in mapped system 91
.WRB (write a series of disk file blocks) 68
.WRCMN (write a message to the other program) 166
.WREBL (remove write protection from protected
  memory area) 104
Write-protecting memory (Figure) 103
.WRL (write a line) 64
.WROPR (write an operator message) 167
.WRPR (protect memory area from modification) 104
.WRR (write random record) 68
.WRS (write sequential) 66

X

.XMT and .XMTW (transmit a message and wait) 129

Y

Y (directory file) 10

Z

ZREL (page zero relocatable memory) 3
ZREL space, to conserve 119

DG OFFICES

NORTH AMERICAN OFFICES

Alabama: Birmingham
Arizona: Phoenix, Tucson
Arkansas: Little Rock
California: Anaheim, El Segundo, Fresno, Los Angeles, Oakland, Palo Alto, Riverside, Sacramento, San Diego, San Francisco, Santa Barbara, Sunnyvale, Van Nuys
Colorado: Colorado Springs, Denver
Connecticut: North Branford, Norwalk
Florida: Ft. Lauderdale, Orlando, Tampa
Georgia: Norcross
Idaho: Boise
Iowa: Bettendorf, Des Moines
Illinois: Arlington Heights, Champaign, Chicago, Peoria, Rockford
Indiana: Indianapolis
Kentucky: Louisville
Louisiana: Baton Rouge, Metairie
Maine: Portland, Westbrook
Maryland: Baltimore
Massachusetts: Cambridge, Framingham, Southboro, Waltham, Wellesley, Westboro, West Springfield, Worcester
Michigan: Grand Rapids, Southfield
Minnesota: Richfield
Missouri: Creve Coeur, Kansas City
Mississippi: Jackson
Montana: Billings
Nebraska: Omaha
Nevada: Reno
New Hampshire: Bedford, Portsmouth
New Jersey: Cherry Hill, Somerset, Wayne
New Mexico: Albuquerque
New York: Buffalo, Lake Success, Latham, Liverpool, Melville, New York City, Rochester, White Plains
North Carolina: Charlotte, Greensboro, Greenville, Raleigh, Research Triangle Park
Ohio: Brooklyn Heights, Cincinnati, Columbus, Dayton
Oklahoma: Oklahoma City, Tulsa
Oregon: Lake Oswego
Pennsylvania: Blue Bell, Lancaster, Philadelphia, Pittsburgh
Rhode Island: Providence
South Carolina: Columbia
Tennessee: Knoxville, Memphis, Nashville
Texas: Austin, Dallas, El Paso, Ft. Worth, Houston, San Antonio
Utah: Salt Lake City
Virginia: McLean, Norfolk, Richmond, Salem
Washington: Bellevue, Richland, Spokane
West Virginia: Charleston
Wisconsin: Brookfield, Grand Chute, Madison

DG-0976

INTERNATIONAL OFFICES

Argentina: Buenos Aires
Australia: Adelaide, Brisbane, Hobart, Melbourne, Newcastle, Perth, Sydney
Austria: Vienna
Belgium: Brussels
Bolivia: La Paz
Brazil: Sao Paulo
Canada: Calgary, Edmonton, Montreal, Ottawa, Quebec, Toronto, Vancouver, Winnipeg
Chile: Santiago
Columbia: Bogata
Costa Rica: San Jose
Denmark: Copenhagen
Ecuador: Quito
Egypt: Cairo
Finland: Helsinki
France: Le Plessis-Robinson, Lille, Lyon, Nantes, Paris, Saint Denis, Strasbourg
Guatemala: Guatemala City
Hong Kong
India: Bombay
Indonesia: Jakarta, Pusat
Ireland: Dublin
Israel: Tel Aviv
Italy: Bologna, Florence, Milan, Padua, Rome, Tourin
Japan: Fukuoka, Hiroshima, Nagoya, Osaka, Tokyo, Tsukuba
Jordan: Amman
Korea: Seoul
Kuwait: Kuwait
Lebanon: Beirut
Malaysia: Kuala Lumpur
Mexico: Mexico City, Monterrey
Morocco: Casablanca
The Netherlands: Amsterdam, Rijswijk
New Zealand: Auckland, Wellington
Nicaragua: Managua
Nigeria: Ibadan, Lagos
Norway: Oslo
Paraguay: Asuncion
Peru: Lima
Philippine Islands: Manila
Portugal: Lisbon
Puerto Rico: Hato Rey
Saudi Arabia: Jeddah, Riyadh
Singapore
South Africa: Cape Town, Durban, Johannesburg, Pretoria
Spain: Barcelona, Bibao, Madrid
Sweden: Gothenburg, Malmo, Stockholm
Switzerland: Lausanne, Zurich
Taiwan: Taipei
Thailand: Bangkok
Turkey: Ankara
United Kingdom: Birmingham, Bristol, Glasgow, Hounslow, London, Manchester
Uruguay: Montevideo
USSR: Espoo
Venezuela: Maracaibo
West Germany: Dusseldorf, Frankfurt, Hamburg, Hannover, Munich, Nuremburg, Stuttgart

CUT ALONG DOTTED LINE

DataGeneral

users group

Installation Membership Form

Name _____ Position _____ Date _____

Company, Organization or School _____

Address _____ City _____ State _____ Zip _____

Telephone: Area Code _____ No. _____ Ext. _____

1. Account Category

- ☐ OEM
- ☐ End User
- ☐ System House
- ☐ Government

2. Hardware

	Qty. Installed	Qty. On Order
M/600		
MV/Series ECLIPSE®		
Commercial ECLIPSE		
Scientific ECLIPSE		
Array Processors		
CS Series		
NOVA®4 Family		
Other NOVAs		
microNOVA® Family		
MPT Family		

Other _____
(Specify) _____

3. Software

- ☐ AOS
- ☐ AOS/VS
- ☐ AOS/RT32
- ☐ MP/OS
- ☐ MP/AOS
- ☐ RDOS
- ☐ DOS
- ☐ RTOS
- ☐ Other

Specify _____

4. Languages

- ☐ ALGOL
- ☐ DG/L
- ☐ COBOL
- ☐ Interactive COBOL
- ☐ PASCAL
- ☐ Business BASIC
- ☐ BASIC
- ☐ Assembler
- ☐ FORTRAN 77
- ☐ FORTRAN 5
- ☐ RPG II
- ☐ PL/1
- ☐ APL
- ☐ Other

Specify _____

5. Mode of Operation

- ☐ Batch (Central)
- ☐ Batch (Via RJE)
- ☐ On-Line Interactive

6. Communication

- ☐ HASP
- ☐ HASP II
- ☐ RJE80
- ☐ RCX 70
- ☐ RSTCP
- ☐ 4025
- ☐ X.25
- ☐ SAM
- ☐ CAM
- ☐ XODIAC™
- ☐ DG/SNA
- ☐ 3270
- ☐ Other

Specify _____

7. Application Description

○ _____

8. Purchase

From whom was your machine(s) purchased?

- ☐ **Data General Corp.**
- ☐ Other

Specify _____

9. Users Group

Are you interested in joining a special interest or regional Data General Users Group?

○ _____

DataGeneral

Data General Corporation, Westboro, Massachusetts 01580, (617) 366-8911

FOLD

TAPE

FOLD

TAPE

FOLD

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 26 SOUTHBORO, MA. 01772

Postage will be paid by addressee:

Data General

ATTN: Users Group Coordinator (C-228)
4400 Computer Drive
Westboro, MA 01581

CUT ALONG DOTTED LINE

ISD User Documentation Remarks Form

Your Name _____

Your Title _____

Company _____

Street _____

City _____ State _____ Zip _____

We wrote this book for you, and we made certain assumptions about who you are and how you would use it. Your comments will help us correct our assumptions and improve the manual. Please take a few minutes to respond. Thank you.

Manual Title *RDOS System Reference* Manual No. *093-400027-00*

Who are you?

- ☐ EDP Manager
- ☐ Senior Systems Analyst
- ☐ Other _____
- ☐ Analyst/Programmer
- ☐ Operator

What programming language(s) do you use? _____

How do you use this manual? *(List in order: 1 = Primary Use)*

- ____ Introduction to the product
- ____ Reference
- ____ Other _____
- _____ Tutorial Text
- _____ Operating Guide

About the manual:	Yes	Somewhat	No
Is it easy to read?	☐	☐	☐
Is it easy to understand?	☐	☐	☐
Are the topics logically organized?	☐	☐	☐
Is the technical information accurate?	☐	☐	☐
Can you easily find what you want?	☐	☐	☐
Does it tell you everything you need to know?	☐	☐	☐
Do the illustrations help you?	☐	☐	☐

If you have any comments on the software itself, please contact your Data General Systems Engineer.
If you wish to order manuals, see your Data General Sales Representative.

Remarks:

Date _____

FOLD

TAPE

FOLD

TAPE

FOLD

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 26 SOUTHBORO, MA. 01772

Postage will be paid by addressee:

Data General

User Documentation, M.S. E-111

4400 Computer Drive

Westborough, Massachusetts 01581

Data General

Data General Corporation, Westboro, Massachusetts 01580

093-400027-00