August 1978

This manual describes the VAX/VMS system services. It provides coding conventions, examples of how to use system services, and detailed reference information on the arguments required by each system service.

VAX/VMS
System Services
Reference Manual
Order No. AA-D018A-TE

SUPERSESSION/UPDATE INFORMATION: This is a new document for this release.
OPERATING SYSTEM AND VERSION: VAX/VMS V01
SOFTWARE VERSION: VAX/VMS V01

To order additional copies of this document, contact the Software Distribution Center, Digital Equipment Corporation, Maynard, Massachusetts 01754

digital equipment corporation • maynard, massachusetts
The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may only be used or copied in accordance with the terms of such license.

No responsibility is assumed for the use or reliability of software on equipment that is not supplied by DIGITAL or its affiliated companies.

Copyright © 1978 by Digital Equipment Corporation

The postage-prepaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist us in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

DIGITAL
DEC
PDP
DECUS
UNIBUS
COMPUTER LABS
COMTEX
DDT
DECCOMM
ASSIST-11
VAX
DECnet

DECsysterm-10
DECTape
DIBOL
EDUSYSTEM
FLIP CHIP
FOCAL
INDAC
LAB-8
DECSYSTEM-20
RTS-8
VMS
IAS

MASSBUS
OMNIBUS
OS/8
PHA
RSTS
RSX
TYPESET-8
TYPESET-11
TMS-11
ITPS-10
SBI
**CONTENTS**

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preface</td>
<td></td>
<td></td>
<td>vii</td>
</tr>
<tr>
<td>Chapter 1</td>
<td>1.1</td>
<td>WHO CAN USE SYSTEM SERVICES: PRIVILEGE AND PROTECTION</td>
<td>1-1</td>
</tr>
<tr>
<td>Chapter 1</td>
<td>1.2</td>
<td>SUMMARY OF VAX/VMS SYSTEM SERVICES</td>
<td>1-3</td>
</tr>
<tr>
<td>Chapter 2</td>
<td>2.1</td>
<td>MACRO CODING</td>
<td>2-2</td>
</tr>
<tr>
<td>Chapter 2</td>
<td>2.2</td>
<td>FORTRAN CODING</td>
<td>2-14</td>
</tr>
<tr>
<td>Chapter 3</td>
<td>3.1</td>
<td>EVENT FLAG SERVICES</td>
<td>3-4</td>
</tr>
<tr>
<td>Chapter 3</td>
<td>3.2</td>
<td>AST (ASYNCHRONOUS SYSTEM TRAP) SERVICES</td>
<td>3-10</td>
</tr>
<tr>
<td>Chapter 3</td>
<td>3.3</td>
<td>LOGICAL NAME SERVICES</td>
<td>3-15</td>
</tr>
<tr>
<td>Chapter 3</td>
<td>3.4</td>
<td>INPUT/OUTPUT SERVICES</td>
<td>3-21</td>
</tr>
<tr>
<td>Chapter 3</td>
<td>3.5</td>
<td>PROCESS CONTROL SERVICES</td>
<td>3-37</td>
</tr>
<tr>
<td>Chapter 3</td>
<td>3.6</td>
<td>TIMER AND TIME CONVERSION SERVICES</td>
<td>3-56</td>
</tr>
<tr>
<td>Chapter 3</td>
<td>3.7</td>
<td>CONDITION HANDLING SERVICES</td>
<td>3-63</td>
</tr>
<tr>
<td>Chapter 3</td>
<td>3.8</td>
<td>MEMORY MANAGEMENT SERVICES</td>
<td>3-77</td>
</tr>
<tr>
<td>Chapter 4</td>
<td>4.1</td>
<td>$ADJUSTK - ADJUST OUTER MODE STACK POINTER</td>
<td>4-3</td>
</tr>
<tr>
<td>Chapter 4</td>
<td>4.2</td>
<td>$ADJWSL - ADJUST WORKING SET LIMIT</td>
<td>4-5</td>
</tr>
<tr>
<td>Chapter 4</td>
<td>4.3</td>
<td>$ALLOC - ALLOCATE DEVICE</td>
<td>4-6</td>
</tr>
<tr>
<td>Chapter 4</td>
<td>4.4</td>
<td>$ASCPC - ASSOCIATE COMMON EVENT FLAG CLUSTER</td>
<td>4-8</td>
</tr>
<tr>
<td>Chapter 4</td>
<td>4.5</td>
<td>$ASTIM - CONVERT BINARY TIME TO ASCII STRING</td>
<td>4-10</td>
</tr>
<tr>
<td>Chapter 4</td>
<td>4.6</td>
<td>$ASSIGN - ASSIGN I/O CHANNEL</td>
<td>4-12</td>
</tr>
<tr>
<td>Chapter 4</td>
<td>4.7</td>
<td>$BINTIM - CONVERT ASCII STRING TO BINARY TIME</td>
<td>4-15</td>
</tr>
<tr>
<td>Chapter 4</td>
<td>4.8</td>
<td>$BRDCT - BROADCAST</td>
<td>4-17</td>
</tr>
<tr>
<td>Chapter 4</td>
<td>4.9</td>
<td>$CANCEL - CANCEL I/O ON CHANNEL</td>
<td>4-19</td>
</tr>
<tr>
<td>Chapter 4</td>
<td>4.10</td>
<td>$CANEXH - CANCEL EXIT HANDLER</td>
<td>4-21</td>
</tr>
<tr>
<td>Chapter 4</td>
<td>4.11</td>
<td>$CANTIM - CANCEL TIMER REQUEST</td>
<td>4-22</td>
</tr>
<tr>
<td>Chapter 4</td>
<td>4.12</td>
<td>$CANWAK - CANCEL WAKEUP</td>
<td>4-23</td>
</tr>
<tr>
<td>Chapter 4</td>
<td>4.13</td>
<td>$CLREF - CLEAR EVENT FLAG</td>
<td>4-25</td>
</tr>
<tr>
<td>Chapter 4</td>
<td>4.14</td>
<td>$CMEXEC - CHANGE TO EXECUTIVE MODE</td>
<td>4-26</td>
</tr>
<tr>
<td>Chapter 4</td>
<td>4.15</td>
<td>$CMKRN - CHANGE TO KERNEL MODE</td>
<td>4-27</td>
</tr>
<tr>
<td>Chapter 4</td>
<td>4.16</td>
<td>$CNTREG - CONTRACT PROGRAM/CONTROL REGION</td>
<td>4-28</td>
</tr>
<tr>
<td>Chapter 4</td>
<td>4.17</td>
<td>$CRELOG - CREATE LOGICAL NAME</td>
<td>4-30</td>
</tr>
<tr>
<td>Chapter 4</td>
<td>4.18</td>
<td>$CREMBX - CREATE MAILBOX AND ASSIGN CHANNEL</td>
<td>4-32</td>
</tr>
<tr>
<td>Chapter 4</td>
<td>4.19</td>
<td>$CREPRC - CREATE PROCESS</td>
<td>4-35</td>
</tr>
<tr>
<td>Chapter 4</td>
<td>4.20</td>
<td>$CRETA - CREATE VIRTUAL ADDRESS SPACE</td>
<td>4-44</td>
</tr>
<tr>
<td>Chapter 4</td>
<td>4.21</td>
<td>$CRMPSC - CREATE AND MAP SECTION</td>
<td>4-46</td>
</tr>
<tr>
<td>Chapter 4</td>
<td>4.22</td>
<td>$DACEPC - DISSOCIATE COMMON EVENT FLAG CLUSTER</td>
<td>4-52</td>
</tr>
<tr>
<td>Chapter 4</td>
<td>4.23</td>
<td>$DALOC - DEALLOCATE DEVICE</td>
<td>4-53</td>
</tr>
<tr>
<td>Chapter 4</td>
<td>4.24</td>
<td>$DASSGN - DEASSIGN I/O CHANNEL</td>
<td>4-55</td>
</tr>
<tr>
<td>Page</td>
<td>CONTENTS (Cont.)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4-57</td>
<td>4.25 $DCLAST - DECLARE AST</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4-58</td>
<td>4.26 $DCLCMH - DECLARE CHANGE MODE OR COMPATIBILITY MODE HANDLER</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4-60</td>
<td>4.27 $DCLGEXH - DECLARE EXIT HANDLER</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4-62</td>
<td>4.28 $DELOG - DELETE LOGICAL NAME</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4-64</td>
<td>4.29 $DELMBX - DELETE MAILBOX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4-66</td>
<td>4.30 $DELPRC - DELETE PROCESS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4-68</td>
<td>4.31 $DELTVA - DELETE VIRTUAL ADDRESS SPACE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4-70</td>
<td>4.32 $DGLSE - DELETE GLOBAL SECTION</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4-72</td>
<td>4.33 $DLCFCGC - DELETE COMMON EVENT FLAG CLUSTER</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4-73</td>
<td>4.34 $EXIT - EXIT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4-74</td>
<td>4.35 $EXPREG - EXPAND PROGRAM/CONTROL REGION</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4-76</td>
<td>4.36 $FAO - FORMATTED ASCII OUTPUT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4-90</td>
<td>4.37 $FORCEX - FORCE EXIT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4-92</td>
<td>4.38 $GETCHN - GET I/O CHANNEL INFORMATION</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4-95</td>
<td>4.39 $GETDEV - GET I/O DEVICE INFORMATION</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4-97</td>
<td>4.40 $GETJPI - GET JOB/PROCESS INFORMATION</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4-102</td>
<td>4.41 $GETMSG - GET MESSAGE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4-104</td>
<td>4.42 $GETTIM - GET TIME</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4-105</td>
<td>4.43 $SHIBER - HIBERNATE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4-106</td>
<td>4.44 $INPUT - QUEUE INPUT REQUEST AND WAIT FOR EVENT FLAG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4-107</td>
<td>4.45 $LCKPAG - LOCK PAGES IN MEMORY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4-109</td>
<td>4.46 $LKWSET - LOCK PAGES IN WORKING SET</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4-111</td>
<td>4.47 $MGBLSC - MAP GLOBAL SECTION</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4-114</td>
<td>4.48 $SNUMTIM - CONVERT BINARY TIME TO NUMERIC TIME</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4-116</td>
<td>4.49 $OUTPUT - QUEUE OUTPUT REQUEST AND WAIT FOR EVENT FLAG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4-117</td>
<td>4.50 $PURGWS - PURGE WORKING SET</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4-118</td>
<td>4.51 $PUTMSG - PUT MESSAGE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4-124</td>
<td>4.52 $QIO - QUEUE I/O REQUEST</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4-127</td>
<td>4.53 $QIOI - QUEUE I/O REQUEST AND WAIT FOR EVENT FLAG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4-128</td>
<td>4.54 $READIF - READ EVENT FLAGS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4-129</td>
<td>4.55 $RESUME - RESUME PROCESS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4-131</td>
<td>4.56 $SCHDKW - SCHEDULE WAKEUP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4-133</td>
<td>4.57 $SETAST - SET AST ENABLE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4-134</td>
<td>4.58 $SETFIF - SET EVENT FLAG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4-135</td>
<td>4.59 $SETEXV - SET EXCEPTION VECTOR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4-137</td>
<td>4.60 $SETIMR - SET TIMER</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4-139</td>
<td>4.61 $SEPRA - SET POWER RECOVERY AST</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4-140</td>
<td>4.62 $SETPRI - SET PRIORITY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4-142</td>
<td>4.63 $SETPRN - SET PROCESS NAME</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4-143</td>
<td>4.64 $SETPRT - SET PROTECTION ON PAGES</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4-145</td>
<td>4.65 $SETRWM - SET RESOURCE WAIT MODE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4-146</td>
<td>4.66 $SETSFM - SET SYSTEM SERVICE FAILURE EXCEPTION MODE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4-147</td>
<td>4.67 $SETSWM - SET PROCESS SWAP MODE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4-148</td>
<td>4.68 $SENDACC - SEND MESSAGE TO ACCOUNTING MANAGER</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4-153</td>
<td>4.69 $SNDERR - SEND MESSAGE TO ERROR LOGGER</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4-154</td>
<td>4.70 $SNDOPR - SEND MESSAGE TO OPERATOR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4-159</td>
<td>4.71 $SNDSCMB - SEND MESSAGE TO SYMBIONT MANAGER</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4-169</td>
<td>4.72 $SUSPEND - SUSPEND PROCESS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4-171</td>
<td>4.73 $TRNLOG - TRANSLATE LOGICAL NAME</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4-173</td>
<td>4.74 $ULKPA - UNLOCK PAGES FROM MEMORY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4-175</td>
<td>4.75 $ULNSET - UNLOCK PAGES FROM WORKING SET</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4-177</td>
<td>4.76 $UNWIND - UNWIND CALL STACK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
CONTENTS (Cont.)

4.77 SUPDSEC - UPDATE SECTION FILE ON DISK 4-179
4.78 SWATFR - WAIT FOR SINGLE EVENT FLAG 4-182
4.79 SWAKE - WAKE 4-183
4.80 SWFLAND - WAIT FOR LOGICAL AND OF EVENT FLAGS 4-185
4.81 SWFLOR - WAIT FOR LOGICAL OR OF EVENT FLAGS 4-186

APPENDIX A SYSTEM SYMBOLIC DEFINITION MACROS A-1

A.1 USING SYSTEM SYMBOLS A-2
A.2 $IODEF MACRO - SYMBOLIC NAMES FOR I/O FUNCTION CODES A-2
A.3 $MSGDEF MACRO - SYMBOLIC NAMES FOR SYSTEM MAILBOX MESSAGES A-6
A.4 $PRDEF MACRO - SYMBOLIC NAMES FOR PROCESSOR REGISTERS A-7
A.5 $PRTDEF - HARDWARE PROTECTION CODE DEFINITIONS A-7
A.6 $PSLDEF MACRO - PROCESSOR STATUS LONGWORD SYMBOL DEFINITIONS A-8
A.7 $SSDEF MACRO - SYMBOLIC NAMES FOR SYSTEM STATUS CODES A-8

APPENDIX B PROGRAM EXAMPLES B-1

APPENDIX C QUICK REFERENCE SUMMARY OF SYSTEM SERVICES C-1

C.1 MACRO FORMS C-1
C.2 FORTRAN FORMS C-2
C.3 SYSTEM SERVICE MACROS C-3

INDEX Index-1

FIGURES

FIGURE 1 How to Use This Book viii
3-1 FORTRAN Interpretation of MACRO Examples 3-2
3-2 Using Local Event Flags 3-6
3-3 Example of a Common Event Flag Cluster 3-9
3-4 Example of an AST 3-11
3-5 The AST Service Routine 3-14
3-6 Logical Name Table Entries 3-17
3-7 Synchronizing I/O Completion 3-23
3-8 Example of Terminal Input and Output 3-26
3-9 Device Allocation and Channel Assignment 3-29
3-10 Example of Using Formatted ASCII Output Program 3-32
3-11 Mailbox Creation and I/O 3-34
3-12 Defining Input and Output Streams for a Subprocess 3-39
3-13 Process Hibernation 3-46
3-14 Example of an Exit Handler 3-50
3-15 Image Exit and Process Deletion 3-52
3-16 Using a Termination Mailbox 3-54
3-17 Timer Requests 3-59
3-18 Search of Stack for Condition Handler 3-66
3-19 Argument List and Arrays Passed to Condition Handler 3-67
CONTENTS (Cont.)

FIGURES (Cont.)

FIGURE 3-20 Example of Condition Handling Routines 3-72
3-21 Unwinding the Call Stack 3-75
3-22 Layout of Process Virtual Address Space 3-78
3-23 Creating and Mapping a Private Section 3-86
3-24 Creating and Mapping a Global Section 3-88
4-1 Format of Numeric Time Buffer 4-114

TABLES

TABLE 1-1 Event Flag Services 1-6
1-2 AST (Asynchronous System Trap) Services 1-7
1-3 Logical Name Services 1-7
1-4 Input/Output Services 1-8
1-5 Process Control Services 1-10
1-6 Timer and Time Conversion Services 1-12
1-7 Condition Handling Services 1-13
1-8 Memory Management Services 1-14
1-9 Change Mode Services 1-15
2-1 FORTRAN Arguments for System Services 2-17
3-1 Summary of Event Flag and Cluster Numbers 3-5
3-2 Default Device Names for I/O Services 3-31
3-3 Process Identification 3-43
3-4 Process Hibernation and Suspension 3-45
3-5 Summary of Exception Conditions 3-70
3-6 Sample Virtual Address Arrays 3-80
4-1 Arguments for the $CRMPSCE System Service 4-49
4-2 Summary of FAO Directives 4-80
4-3 How FAO Determines Output Field Lengths and Fill Characters 4-82
4-4 Item Codes for Job/Process Information 4-100
4-5 Format of Accounting Log File Records 4-151
4-6 Request Types for Symbiont Manager Messages 4-163
4-7 Options for Symbiont Manager Messages 4-165
PREFACE

This manual provides users of the VAX/VMS operating system with detailed usage and reference information on the system services.

VAX/VMS system services can be used only in programs written in languages that produce native code for the VAX-11/780 hardware. These languages are:

VAX-11 FORTRAN IV-PLUS
VAX-11 MACRO

INTENDED AUDIENCE

This manual is intended for system and application programmers who are already familiar with VAX/VMS system concepts. For an overview of the operating system and an introduction to some of the concepts used in system services, see the VAX/VMS Summary Description.

STRUCTURE OF THIS DOCUMENT

This manual is organized into four chapters and three appendixes, as follows:

- Chapter 1 contains introductory information: it presents overviews of the various categories of system services, and summarizes the services in each category.
- Chapter 2 describes the conventions used to code calls to system services. It discusses the macro forms for coding in VAX-11 MACRO, and tells how to call the services from a program written in VAX-11 FORTRAN IV-PLUS.
- Chapter 3 contains usage information intended to guide new users in understanding how the system services work and how to use them.
- Chapter 4 provides detailed reference information on each system service. The descriptions are presented in alphabetical order, for ease of reference.
- Appendix A lists the system-provided macro instructions that define symbolic names for frequently used system constants.
Appendix B contains sample programs that use various system services.

Appendix C summarizes the system-service formats, for easy reference.

See Figure 1 for an illustration of how to use this book.

Figure 1  How to Use This Book
ASSOCIATED DOCUMENTS

The following documents are prerequisite for

- All Users:
  VAX/VMS Summary Description

- MACRO Programmers:
  VAX-11 MACRO User's Guide

- FORTRAN Programmers:
  VAX-11 FORTRAN IV-PLUS User's Guide

The following documents, which are referred to in this manual, may also be useful:

- VAX/VMS Command Language User's Guide
- Introduction to VAX-11 Record Management Services
- VAX-11 Record Management Services Reference Manual
- VAX/VMS I/O User's Guide
- DECnet-VAX User's Guide

For a complete list of VAX-11 documents, including descriptions of each, see the VAX-11 Information Directory.

CONVENTIONS USED IN THIS DOCUMENT

The following syntactical conventions are used in this manual:

- Brackets ([ ]) in system service descriptions indicate optional arguments.

- Horizontal ellipses (...) indicate:
  (1) when shown in the format of a system service call, that additional optional arguments have been omitted;
  (2) when shown in an example, that additional arguments required by a service but not pertinent to the example are not shown.

- Vertical ellipses in coding examples indicate that lines of code not pertinent to the example are not shown. For example:

  ...

- Uppercase letters in a system service format show keywords that must be entered as shown; lowercase letters show variable data.
CHAPTER 1

INTRODUCTION TO SYSTEM SERVICES

System services are procedures that the VAX/VMS operating system uses to control resources available to processes; to provide for communication among processes; and to perform basic operating system functions, such as the coordination of input/output operations.

Although most system services are used primarily by the operating system itself on behalf of logged-in users, many are available for general use and provide techniques that can be used in application programs. For example, when you log into the system, the Create Process system service is called to create a process on your behalf. You may, in turn, code a program that calls the Create Process system service to create a subprocess.

1.1 WHO CAN USE SYSTEM SERVICES: PRIVILEGE AND PROTECTION

Many system services are available and suitable for application programs, but the use of some services must be restricted to protect:

- The performance of the system
- The integrity of user processes

For example, because the creation of permanent mailboxes uses system dynamic memory, the unrestricted use of permanent mailboxes could decrease the amount of memory available to other users. Therefore, the ability to create permanent mailboxes is controlled: a user must be specifically assigned the privilege to use the Create Mailbox system service to create a permanent mailbox.

The various controls and restrictions applied to system service usage are described below. The tables in Section 1.2 that summarize the system services note any restrictions on the use of specific services.

1.1.1 User Privileges and Resource Quotas

The system manager, who maintains the user authorization file for the system, grants privileges to use protected system services. The user authorization file contains, in addition to profile information on each user, a list of specific user privileges and resource quotas.

When you log into the system, the privileges and quotas you have been assigned are associated with the process created on your behalf. These privileges and quotas are applied to every image that the process executes.
INTRODUCTION TO SYSTEM SERVICES

When an image issues a call to a system service that is protected by privilege, the privilege list is checked. If you have been granted the specific privilege required, the image is allowed to execute the system service; otherwise, a status code indicating an error is returned.

When a system service that uses a resource controlled by a quota is called, the process's quota for that resource is checked. If the process has exceeded its quota, or if it has no quota allotment, an error status code may be returned. In some cases, the process may be placed in a wait state until the resource becomes available; see Section 2.1.5.4, "Special Return Conditions."

1.1.2 Control by Group Association

Some system services provide techniques for coordinating and synchronizing the execution of different processes. These services require cooperating processes to be in the same group; that is, the group fields in the user identification codes (UICs) for the processes must match.

For example, event flags are used to post the occurrence of events in a program and can be shared among cooperating processes. However, the processes that share a cluster of event flags must be in the same group.

1.1.3 Protection by Access Mode

A process can execute at any one of four access modes: user, supervisor, executive, or kernel. The access modes determine a process's ability to access pages of virtual memory. Each page has a protection code associated with it, specifying the type of access -- read, write, or no access -- allowed for each mode. The VAX-11/780 Architecture Handbook provides additional information on access modes.

For the most part, user-written programs execute in user mode; system programs executing at the user's request (system services, for example) may execute at one of the other three, more privileged, access modes.

In some system service calls, the access mode of the caller is checked. For example, when a process tries to cancel timer requests, it can cancel only those requests that were issued from the same or less privileged access modes. For example, a process executing in user mode cannot cancel a timer request made from supervisor, executive, or kernel mode, which are more privileged access modes.
1.2 SUMMARY OF VAX/VMS SYSTEM SERVICES

The following sections summarize the VAX/VMS system services in functional groups, with tables listing the services that belong in each group. Each table lists:

- The full name of the service and the short, macro name by which it is alphabetized in this book.
- The functions performed by the service, with distinctions based on privilege (where applicable).
- Restrictions on the use of the service, if any. This column is keyed as follows:
  
  None indicates that no restriction is placed on the use of the service for this function.

  xxx privilege indicates the specific user privilege that is required to use the service for the requested function.

  yyy quota indicates the specific resource quota that is required to use the service for the requested function.

  Access mode indicates that this service uses the access mode of the caller to determine whether the caller can execute the function requested.

  UIC protection indicates that this service may restrict access based on the caller's UIC.

For detailed information about a restriction applied to any specific service, see that service's description in Chapter 4.

Chapter 3 provides additional information, including examples, on the services listed in Tables 1-1 through 1-8.

1.2.1 Event Flag Services

A process can use event flags to synchronize sequences of operations in a program. Event flag services clear, set, and read event flags, and place a process in a wait state pending the setting of an event flag or flags.

Table 1-1 lists the event flag services.

1.2.2 AST (Asynchronous System Trap) Services

Process execution can be interrupted by events (such as I/O completion) for the execution of designated subroutines. These software interrupts are called asynchronous system traps (ASTs) because they occur asynchronously to process execution. System services are provided so that a process can control the handling of ASTs.

Table 1-2 lists the AST services.
1.2.3 Logical Name Services

Logical name services provide a generalized technique for maintaining and accessing character string logical name and equivalence name pairs. Logical names can provide device-independence for system and application program input and output operations.

Table 1-3 lists the logical name services.

1.2.4 Input/Output Services

I/O services perform input and output operations directly, rather than through the file handling services of the VAX-11 Record Management Services (RMS). I/O services:

- Perform logical and virtual input/output operations
- Format output lines converting binary numeric values to ASCII strings and substituting variable data in ASCII strings
- Create mailboxes for interprocess communication
- Perform network operations
- Queue messages to system processes

Table 1-4 lists the I/O services. The following manuals provide additional information on aspects of input/output operations not covered in this manual:

- Introduction to VAX-11 Record Management Services
- VAX-11 Record Management Services Reference Manual
- VAX/VMS I/O User's Guide
- DECnet-VAX User's Guide

1.2.5 Process Control Services

Process control services allow you to create, delete, and control the execution of processes.

Table 1-5 lists the process control services.

1.2.6 Timer and Time Conversion Services

Timer services schedule program events for a particular time of day, or for after a specified interval of time has elapsed. The time conversion services provide a way to obtain and format binary time values for use with the timer services.

Table 1-6 lists the timer and time conversion services.
1.2.7 Condition Handling Services

Condition handlers are procedures that can be designated to receive control when a hardware or software exception condition occurs during image execution. Condition handling services designate condition handlers for special purposes.

Table 1-7 lists the condition handling services.

1.2.8 Memory Management Services

Memory management services provide ways to use the virtual address space available to a program. Included are services that:

- Allow an image to increase or decrease the amount of virtual memory available
- Control the paging and swapping of virtual memory
- Create and access in memory files that contain shareable code or data

Table 1-8 lists the memory management services.

1.2.9 Change Mode Services

Change mode services alter the access mode of a process to a more privileged mode to execute particular routines. These services are used primarily by the operating system.

Table 1-9 lists the change mode services.
## Table 1-1
### Event Flag Services

<table>
<thead>
<tr>
<th>Service Name</th>
<th>Function(s)</th>
<th>Restriction(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Associate Common Event Flag Cluster ($ASCEFC)</td>
<td>Creates a temporary common event flag cluster</td>
<td>TQELM quota</td>
</tr>
<tr>
<td></td>
<td>Creates a permanent common event flag cluster</td>
<td>PRMCEB privilege</td>
</tr>
<tr>
<td></td>
<td>Establishes association with an existing common event flag cluster</td>
<td>Group association</td>
</tr>
<tr>
<td>Disassociate Common Event Flag Cluster ($DACEFC)</td>
<td>Cancels association with a common event flag cluster</td>
<td>None</td>
</tr>
<tr>
<td>Delete Common Event Flag Cluster ($DLCEFC)</td>
<td>Marks a permanent common event flag cluster for deletion</td>
<td>PRMCEB privilege, Group association</td>
</tr>
<tr>
<td>Set Event Flag ($SETEF)</td>
<td>Turns on an event flag in a process-local event flag cluster</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>Turns on an event flag in a common event flag cluster</td>
<td>Group association</td>
</tr>
<tr>
<td>Clear Event Flag ($CLEFEP)</td>
<td>Turns off an event flag in a process-local event flag cluster</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>Turns off an event flag in a common event flag cluster</td>
<td>Group association</td>
</tr>
<tr>
<td>Read Event Flags ($READEF)</td>
<td>Returns the status of all event flags in a process-local event flag cluster</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>Returns the status of all event flags in a common event flag cluster</td>
<td>Group association</td>
</tr>
<tr>
<td>Wait for Single Event Flag ($WAITFR)</td>
<td>Places the current process in a wait state pending the setting of an event flag in a process-local event flag cluster</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>Places the current process in a wait state pending the setting of an event flag in a common event flag cluster</td>
<td>Group association</td>
</tr>
<tr>
<td>Wait for Logical OR of Event Flags ($WFLOR)</td>
<td>Places the current process in a wait state pending the setting of any one of a specified set of flags in a process-local event flag cluster</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>Places the current process in a wait state pending the setting of any one of a specified set of flags in a common event flag cluster</td>
<td>Group association</td>
</tr>
<tr>
<td>Wait for Logical AND of Event Flags ($WFLAND)</td>
<td>Places the current process in a wait state pending the setting of all specified flags in a process-local event flag cluster</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>Places the current process in a wait state pending the setting of all specified flags in a common event flag cluster</td>
<td>Group association</td>
</tr>
</tbody>
</table>

1 For an explanation of the terms used in this column, see Page 1-3.
# INTRODUCTION TO SYSTEM SERVICES

## Table 1-2
### AST (Asynchronous System Trap) Services

<table>
<thead>
<tr>
<th>Service Name</th>
<th>Function(s)</th>
<th>Restriction(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set AST Enable ($$SETAST)</td>
<td>Enables or disables the delivery of ASTs</td>
<td>None</td>
</tr>
<tr>
<td>Declare AST ($$DECLAST)</td>
<td>Queues an AST for delivery</td>
<td>ASTLM quota</td>
</tr>
<tr>
<td>Set Power Recovery AST ($$SETPRA)</td>
<td>Establishes AST routine to receive control following power recovery condition</td>
<td>ASTLM quota</td>
</tr>
</tbody>
</table>

1 For an explanation of the terms used in this column, see Page 1-3.

## Table 1-3
### Logical Name Services

<table>
<thead>
<tr>
<th>Service Name</th>
<th>Function(s)</th>
<th>Restriction(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create Logical Name ($$CRELOG)</td>
<td>Places logical name/equivalence name pair in process logical name table</td>
<td>Access mode</td>
</tr>
</tbody>
</table>
|                                    | Places logical name/equivalence name pair in group logical name table | GRPNAM privilege
|                                    | Places logical name/equivalence name pair in system logical name table | SYSNAM privilege |
| Delete Logical Name ($$DELLOG)      | Removes logical name/equivalence name pair from process logical name table | None           |
|                                    | Removes logical name/equivalence name pair from group logical name table | GRPNAM privilege
|                                    | Removes logical name/equivalence name pair from system logical name table | SYSNAM privilege |
| Translate Logical Name ($$TRNLOG)   | Searches logical name tables for a specified logical name and return its equivalence name when the first match is found | None           |

1 For an explanation of the terms used in this column, see Page 1-3.
### Table 1-4
Input/Output Services

<table>
<thead>
<tr>
<th>Service Name</th>
<th>Function(s)</th>
<th>Restriction(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assign I/O Channel ($ASSIGN)</td>
<td>Establishes a path for an I/O request</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>Establishes a path for network operations</td>
<td>NETMBX privilege</td>
</tr>
<tr>
<td>Deassign I/O Channel ($DASSIGN)</td>
<td>Releases linkage for an I/O path</td>
<td>Access mode</td>
</tr>
<tr>
<td></td>
<td>Releases a path from the network</td>
<td>NETMBX privilege</td>
</tr>
<tr>
<td>Queue I/O Request ($QIO)</td>
<td>Initiates an input or output operation</td>
<td>Access mode^2</td>
</tr>
<tr>
<td>Queue I/O Request and Wait for Event Flag ($QIOW)</td>
<td>Initiates an input or output operation and causes the process to wait until it is complete before continuing execution</td>
<td>Access mode^2</td>
</tr>
<tr>
<td>INPUT</td>
<td>Initiates virtual input operation and waits for completion</td>
<td>Access mode^2</td>
</tr>
<tr>
<td>OUTPUT</td>
<td>Initiates virtual output operation and waits for completion</td>
<td>Access mode^2</td>
</tr>
<tr>
<td>Formatted ASCII Output ($FASO)</td>
<td>Performs ASCII string substitution, and converts numeric data to ASCII representation and substitutes in output</td>
<td>None</td>
</tr>
<tr>
<td>Formatted ASCII Output with List Parameter ($FASPL)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Allocate Device ($ALLOC)</td>
<td>Reserves a device for exclusive use by a process and its subprocesses</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>Reserves a spooled device for exclusive use</td>
<td>ALLSPOOL privilege</td>
</tr>
<tr>
<td>Deallocate Device ($DALLOC)</td>
<td>Relinquishes exclusive use of a device</td>
<td>Access mode</td>
</tr>
<tr>
<td>Get I/O Channel Information ($GETCHN)</td>
<td>Provides information about a device to which an I/O channel has been assigned</td>
<td>Access mode</td>
</tr>
<tr>
<td>Get I/O Device Information ($GETDEV)</td>
<td>Provides information about a physical device</td>
<td>None</td>
</tr>
<tr>
<td>Cancel I/O on Channel ($CANCEL)</td>
<td>Cancels pending I/O requests on a channel</td>
<td>Access mode</td>
</tr>
</tbody>
</table>

1 For an explanation of the terms used in this column, see Page 1-3.

2 Depending on the specific nature of the input or output request, the service may require the PHY_IO, LOG_IO, or MOUNT privileges, or quotas for buffered I/O (B10LM), direct I/O (D10LM), buffer space (BYLM), or AST limit (ASTLM).
## Table 1-4 (Cont.)
Input/Output Services

<table>
<thead>
<tr>
<th>Service Name</th>
<th>Function(s)</th>
<th>Restriction(s)1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create Mailbox and Assign Channel ($CREMBX)</td>
<td>Creates a temporary mailbox</td>
<td>BYTLM/Quota</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TMPMBX privilege</td>
</tr>
<tr>
<td></td>
<td>Creates a permanent mailbox</td>
<td>PRMMBX privilege</td>
</tr>
<tr>
<td>Delete Mailbox ($DELMBX)</td>
<td>Marks a permanent mailbox for deletion</td>
<td>PRMMBX privilege</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Access mode</td>
</tr>
<tr>
<td>Broadcast ($BROADCAST)</td>
<td>Sends a high-priority message to an assigned terminal</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>Sends a high-priority message to a nonassigned terminal or to all terminals</td>
<td>OPER privilege</td>
</tr>
<tr>
<td>Send Message to Accounting Manager ($SSNDACC)</td>
<td>Controls accounting log file activity</td>
<td>OPER privilege</td>
</tr>
<tr>
<td></td>
<td>Writes an arbitrary message to the accounting log file</td>
<td>None</td>
</tr>
<tr>
<td>Send Message to Symbiont Manager ($SSNDSMB)</td>
<td>Requests symbiont manager to initialize, modify, or delete a printer or</td>
<td>OPER privilege</td>
</tr>
<tr>
<td></td>
<td>batch job queue, or a device queue</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Requests symbiont manager to delete or change characteristics of a queued</td>
<td>Group association</td>
</tr>
<tr>
<td></td>
<td>file</td>
<td></td>
</tr>
<tr>
<td>Send Message to Operator ($SSNDOPR)</td>
<td>Writes a message to designated operator(s) terminal(s)</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>Enables or disables an operator's terminal, sends a reply to a user request</td>
<td>OPER privilege</td>
</tr>
<tr>
<td></td>
<td>or initializes the operator's log file</td>
<td></td>
</tr>
<tr>
<td>Send Message to Error Logger ($SSNDRER)</td>
<td>Writes arbitrary data to the system error log file</td>
<td>BUGCHK privilege</td>
</tr>
<tr>
<td>Get Message ($GETMSG)</td>
<td>Returns text of system error message from message file</td>
<td>None</td>
</tr>
<tr>
<td>Put Message ($PUTMSG)</td>
<td>Writes a message to the current output and error devices</td>
<td>None</td>
</tr>
</tbody>
</table>

1 For an explanation of the terms used in this column, see Page 1-3.
# INTRODUCTION TO SYSTEM SERVICES

## Table 1-5

### Process Control Services

<table>
<thead>
<tr>
<th>Service Name</th>
<th>Function(s)</th>
<th>Restriction(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create Process ($CREPFC)</td>
<td>Creates a subprocess</td>
<td>PRCLM quota</td>
</tr>
<tr>
<td></td>
<td>Creates a detached process</td>
<td>DETACH privilege</td>
</tr>
<tr>
<td>Delete Process ($DELPFC)</td>
<td>Deletes the current process or a subprocess</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>Deletes another process in the same group</td>
<td>GROUP privilege</td>
</tr>
<tr>
<td></td>
<td>Deletes any process in the system</td>
<td>Group association</td>
</tr>
<tr>
<td></td>
<td>Makes the current process or a subprocess nonexecutable and unable to receive ASTs until a subsequent resume or delete request</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>Makes another process in the same group nonexecutable and unable to receive ASTs until a subsequent resume or delete request</td>
<td>GROUP privilege</td>
</tr>
<tr>
<td></td>
<td>Makes any process in the system nonexecutable and noninterruptible until a subsequent resume or delete request</td>
<td>Group association</td>
</tr>
<tr>
<td></td>
<td>Makes the current process nonexecutable and unable to receive ASTs until a subsequent resume or delete request</td>
<td>WORLD privilege</td>
</tr>
<tr>
<td>Resume Process ($RESUME)</td>
<td>Restores executability of a suspended process in the same group</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>Restores executability of any suspended process in the system</td>
<td>GROUP privilege</td>
</tr>
<tr>
<td></td>
<td>Restores executability of any suspended process in the system</td>
<td>Group association</td>
</tr>
<tr>
<td>Hibernate ($SHIBER)</td>
<td>Makes the current process dormant but able to receive ASTs until a subsequent wakeup request</td>
<td>None</td>
</tr>
<tr>
<td>Wake ($SWAKE)</td>
<td>Restores executability of the current process or a hibernating subprocess</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>Restores executability of a hibernating process in the same group</td>
<td>GROUP privilege</td>
</tr>
<tr>
<td></td>
<td>Restores executability of any hibernating process in the system</td>
<td>Group association</td>
</tr>
<tr>
<td>Schedule Wakeup ($SCHDWRK)</td>
<td>Wakes a process after a specified time interval or at a specific time ²</td>
<td>WORLD privilege</td>
</tr>
<tr>
<td>Cancel Wakeup ($CANWRK)</td>
<td>Cancels a scheduled wakeup request²</td>
<td>None</td>
</tr>
<tr>
<td>Exit ($EXIT)</td>
<td>Terminates execution of an image and returns to command interpreter</td>
<td>None</td>
</tr>
<tr>
<td>Force Exit ($FORCEX)</td>
<td>Causes image exit for the current process or a subprocess</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>Causes image exit for a process in the same group</td>
<td>GROUP privilege</td>
</tr>
<tr>
<td></td>
<td>Causes image exit for any process in the system</td>
<td>Group association</td>
</tr>
<tr>
<td>Declare Exit Handler ($DCLLEXH)</td>
<td>Designates a routine to receive control when image exits</td>
<td>None</td>
</tr>
<tr>
<td>Cancel Exit Handler ($CANCEXH)</td>
<td>Cancels a previously established exit handling routine</td>
<td>Access mode</td>
</tr>
<tr>
<td>Set Process Name ($SETPRN)</td>
<td>Establishes a text name string to be used to identify the current process</td>
<td>None</td>
</tr>
</tbody>
</table>

¹ For an explanation of the terms used in this column, see Page 1-3.

² Functions performed by these services are listed in detail in Table 1-6.
### Table 1-5 (Cont.)
**Process Control Services**

<table>
<thead>
<tr>
<th>Service Name</th>
<th>Function(s)</th>
<th>Restriction(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Set Priority ($$SETPRI)</strong></td>
<td>Increases the execution priority for any process</td>
<td>ALTPRI privilege</td>
</tr>
<tr>
<td></td>
<td>Changes the execution priority for the current process or a subprocess</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>Changes the execution priority for a process in the same group</td>
<td>GROUP privilege Group association</td>
</tr>
<tr>
<td></td>
<td>Changes the execution priority for any process in the system</td>
<td>WORLD privilege</td>
</tr>
<tr>
<td><strong>Set Resource Wait Mode ($$SETRWM)</strong></td>
<td>Requests wait, or that control be returned immediately, when a system service call cannot be executed because a system resource is not available</td>
<td>None</td>
</tr>
<tr>
<td><strong>Get Job/Process Information ($$GETJPI)</strong></td>
<td>Returns information about the current process</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>Returns information about the current context of other processes in the same group</td>
<td>GROUP privilege Group association</td>
</tr>
<tr>
<td></td>
<td>Returns information about any other process in the system</td>
<td>WORLD privilege</td>
</tr>
</tbody>
</table>

1 For an explanation of the terms used in this column, see Page 1-3.
## Table 1-6
Timer and Time Conversion Services

<table>
<thead>
<tr>
<th>Service Name</th>
<th>Function(s)</th>
<th>Restriction(s)</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get Time ($GETTIM)</td>
<td>Returns the date and time in system format</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>Convert Binary Time to Numeric Time ($NUMTIM)</td>
<td>Converts a date and time from system format to numeric integer values</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>Convert Binary Time to ASCII String ($ASCSTR)</td>
<td>Converts a date and time from system format to an ASCII string</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>Convert ASCII String to Binary Time ($BINTIM)</td>
<td>Converts a date and time in an ASCII string to the system date and time format</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>Set Timer ($SETIMR)</td>
<td>Request setting of an event flag or queueing of an AST based on an absolute or delta time value</td>
<td>TQUELM quota²</td>
<td></td>
</tr>
<tr>
<td>Cancel Timer Request ($CANTIM)</td>
<td>Cancels previously issued timer requests</td>
<td>Access mode</td>
<td></td>
</tr>
<tr>
<td>Schedule Wakeup ($SSCHDWK)</td>
<td>Schedules a wakeup for the current process or a hibernating subprocess</td>
<td>ASTLM quota</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Schedules a wakeup for a hibernating process in the same group</td>
<td>GROUP privilege ASTLM quota Group association</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Schedules a wakeup for any hibernating process in the system</td>
<td>WORLD privilege ASTLM quota</td>
<td></td>
</tr>
<tr>
<td>Cancel Wakeup ($SCANWAK)</td>
<td>Cancels a scheduled wakeup request for the current process or a hibernating subprocess</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Cancels a scheduled wakeup request for a hibernating process</td>
<td>GROUP privilege Group association</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Cancels a scheduled wakeup request for any hibernating process in the system</td>
<td>WORLD privilege</td>
<td></td>
</tr>
</tbody>
</table>

¹ For an explanation of the terms used in this column, see Page 1-3.

² Setting an event flag in a common event flag cluster requires association based on group number; a timer request with an AST requires ASTLM quota.
4.64 $SETPRT - SET PROTECTION ON PAGES

The Set Protection On Pages system service allows an image running in a process to change the protection on a page or range of pages.

Macro Format:

$SETPRT inadr , [retadr] , [acmode] , prot , [prvprt]

High-Level Language Format:

SYSSETPRT(inadr , [retadr] , [acmode] , prot , [prvprt])

inadr
address of a 2-longword array containing the starting and ending virtual addresses of the pages on which protection is to be changed. If the starting and ending virtual addresses are the same, a single page is changed. Only the virtual page number portion of the virtual address is used; the low-order 9 bits are ignored.

retadr
address of a 2-longword array to receive the starting and ending virtual addresses of the pages that had their protection changed.

acmode
access mode on behalf of which the request is being made. The specified access mode is maximized with the access mode of the caller. The resultant access mode must be equal to or more privileged than the access mode of the owner of each page in order to change the protection.

prot
new protection specified in bits 0 through 3 in the format of the hardware page protection. The high-order 28 bits are ignored. Symbolic names defining the protection codes are listed in Appendix A, Section A.5 "$PRTDEF - Hardware Protection Code Definitions."

If the protection is specified as 0, the protection defaults to kernel read-only.

prvprt
address of a byte to receive the protection previously assigned to the last page whose protection was changed. This argument is useful only when protection for a single page is being changed.
SYSTEM SERVICE DESCRIPTIONS
$SETPRT - SET PROTECTION ON PAGES

Return Status:

SS$NORMAL
- Service successfully completed.

SS$ACCVIO
1. The input address array cannot be read, or the output address array or the byte to receive the previous protection cannot be written, by the caller.

2. An attempt was made to change the protection of a nonexistent page.

SS$EXQUOTA
The process exceeded its paging file quota while changing a page in a read-only private section to a read/write page.

SS$IVPROTECT
The specified protection code has a numeric value of 1 or is greater than 15.

SS$LENVIO
A page in the specified range is beyond the end of the program or control region.

SS$NOPRIV
A page in the specified range is in the system address space.

SS$PAGOWNVIO
Page owner violation. An attempt was made to change the protection on a page owned by a more privileged access mode.

Privilege Restrictions:

For pages in global sections, the new protection can alter only the accessibility of the page for modes less privileged than the owner of the page.

Resources Required/Returned:

If a process changes any pages in a private section from read-only to read/write, the service uses the process's paging file quota (PGPLQUOTA).

Note:

If an error occurs while changing page protection, the return array, if requested, indicates the pages that were successfully changed before the error occurred. If no pages have been affected, both longwords in the return address array contain a -1.
### Table 1-7
Condition Handling Services

<table>
<thead>
<tr>
<th>Service Name</th>
<th>Function(s)</th>
<th>Restriction(s)(^1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set Exception Vector ($SETEXV)</td>
<td>Defines condition handlers to receive control in case of hardware- or software-detected exception conditions</td>
<td>Access mode</td>
</tr>
<tr>
<td>Set System Service Failure Exception Mode ($SETSFM)</td>
<td>Requests or disables generation of a software exception condition when a system service call returns an error or severe error</td>
<td>None</td>
</tr>
<tr>
<td>Unwind from Condition Handler Frame ($SUNWIND)</td>
<td>Deletes a specified number of call frames from the call stack following a nonrecoverable exception condition</td>
<td>None</td>
</tr>
<tr>
<td>Declare Change Mode or Compatibility Mode Handler ($DCLCMH)</td>
<td>Designates a routine to receive control when change mode to user instructions are encountered</td>
<td>Access mode</td>
</tr>
<tr>
<td></td>
<td>Designates a routine to receive control when change mode to supervisor instructions are encountered</td>
<td>Access mode</td>
</tr>
<tr>
<td></td>
<td>Designates a routine to receive control when compatibility mode exceptions occur</td>
<td>None</td>
</tr>
</tbody>
</table>

\(^1\) For an explanation of the terms used in this column, see Page 1-3.
### INTRODUCTION TO SYSTEM SERVICES

#### Table 1-8

Memory Management Services

<table>
<thead>
<tr>
<th>Service Name</th>
<th>Function(s)</th>
<th>Restriction(s)¹</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expand Program/Control Region (SEXPREG)</td>
<td>Adds pages at the end of the program or control region</td>
<td>None</td>
</tr>
<tr>
<td>Contract Program/Control Region (SCNTRGREG)</td>
<td>Deletes pages from the end of the program or control region</td>
<td>None</td>
</tr>
<tr>
<td>Create Virtual Address Space ($CRETVA)</td>
<td>Adds pages to the virtual address space available to an image</td>
<td>None</td>
</tr>
<tr>
<td>Delete Virtual Address Space ($DELTA)</td>
<td>Makes a range of virtual addresses unavailable to an image</td>
<td>None</td>
</tr>
<tr>
<td>Create and Map Section ($CRMPSC)</td>
<td>Identifies a disk file as a private section and establishes correspondence between virtual blocks in the file and the process's virtual address space</td>
<td>Access mode</td>
</tr>
<tr>
<td></td>
<td>Identifies a disk file containing shareable code or data as a temporary global section and establishes correspondence between virtual blocks in the file and the process's virtual address space</td>
<td>Access mode</td>
</tr>
<tr>
<td></td>
<td>Identifies a disk file containing shareable code or data as a system global section and establishes correspondence between virtual blocks in the file and the process's virtual address space</td>
<td>PRMGBL privilege Access mode</td>
</tr>
<tr>
<td></td>
<td>Identifies a disk file containing shareable code or data as a system global section and establishes correspondence between virtual blocks in the file and the process's virtual address space</td>
<td>SYSGBL privilege Access mode</td>
</tr>
<tr>
<td>Update Section File on Disk ($SUPDSEC)</td>
<td>Writes modified pages of a private or global section into the section file</td>
<td>Access mode</td>
</tr>
<tr>
<td>Map Global Section ($MGBLSC)</td>
<td>Establishes correspondence between a global section and a process's virtual address space</td>
<td>UIC protection</td>
</tr>
<tr>
<td>Delete Global Section ($DGBLSC)</td>
<td>Marks a permanent global section for deletion</td>
<td>PRMGBL privilege</td>
</tr>
<tr>
<td></td>
<td>Marks a system global section for deletion</td>
<td>SYSGBL privilege Access mode</td>
</tr>
<tr>
<td>Lock Pages in Working Set ($LKWSET)</td>
<td>Specifies that particular pages cannot be paged out of the process's working set</td>
<td>Access mode</td>
</tr>
<tr>
<td>Unlock Pages from Working Set ($ULWSET)</td>
<td>Allows previously locked pages to be paged out of working set</td>
<td>Access mode</td>
</tr>
<tr>
<td>Purge Working Set ($PURGWS)</td>
<td>Removes all pages within a specified range from the current working set</td>
<td>None</td>
</tr>
<tr>
<td>Lock Page in Memory ($LCPAG)</td>
<td>Specifies that particular pages may not be swapped out of memory</td>
<td>User privilege Access mode</td>
</tr>
</tbody>
</table>

¹ For an explanation of the terms used in this column, see Page 1-3.
### Table 1-8 (Cont.)
Memory Management Services

<table>
<thead>
<tr>
<th>Service Name</th>
<th>Function(s)</th>
<th>Restriction(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unlock Page in Memory ($ULRPG)</td>
<td>Allows previously locked pages to be swapped out of memory</td>
<td>User privilege Access mode</td>
</tr>
<tr>
<td>Adjust Working Set Limit ($ADJWSL)</td>
<td>Changes maximum number of pages that the current process can have in its working set</td>
<td>WSQUOTA quota</td>
</tr>
<tr>
<td>Set Protection on Pages ($SETPRT)</td>
<td>Controls access to a range of virtual addresses</td>
<td>Access mode</td>
</tr>
<tr>
<td>Set Process Swap Mode ($SETPSWM)</td>
<td>Controls whether or not the current process can be swapped out of the balance set</td>
<td>PSWAPM privilege</td>
</tr>
</tbody>
</table>

1 For an explanation of the terms used in this column, see Page 1-3.

### Table 1-9
Change Mode Services

<table>
<thead>
<tr>
<th>Service Name</th>
<th>Function(s)</th>
<th>Restriction(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Change to Executive Mode ($CMEXEC)</td>
<td>Executes a specified routine in executive mode</td>
<td>CMEXEC privilege Access mode</td>
</tr>
<tr>
<td>Change to Kernel Mode ($CMKRNLSL)</td>
<td>Executes a specified routine in kernel mode</td>
<td>CMKRNLSL privilege Access mode</td>
</tr>
<tr>
<td>Adjust Outer Mode Stack Pointer ($ADJSTK)</td>
<td>Modifies the current stack pointer for a less privileged access mode</td>
<td>Access mode</td>
</tr>
</tbody>
</table>

1 For an explanation of terms used in this column, see Page 1-3.
CHAPTER 2

CALLING THE SYSTEM SERVICES

System service procedures are called using the standard VAX-11/780 procedure calling conventions. The programming languages that generate VAX-11/780 native mode instructions provide mechanisms for coding the procedure calls. These languages, and supporting documentation, are listed in the Preface.

When you code a system service call, you must supply whatever arguments the service requires.

When the service completes execution, it returns control to the calling program with a return status code. The caller should analyze the status code to determine the success or failure of the service call, so it can alter the flow of execution, if necessary.

This chapter provides all the information you need to code calls to system services.

If you are a VAX-11 MACRO programmer, you should read Section 2.1 for details on how to code the macro instructions that generate system service calls.

If you are a VAX-11 FORTRAN IV-PLUS programmer, you should read Section 2.2 for details on how to code subroutine CALL statements or function references.

Each of these sections also discusses conventions for coding arguments and methods of checking the successful completion of a system service.
2.1 MACRO CODING

System service macros generate argument lists and CALL instructions to call system services. These macros are located in the system library STARLET.MLB; this library is searched automatically for unresolved references when you assemble a source program.


2.1.1 Argument Lists

You can determine the arguments required by a system service from the service description in Chapter 4. The "Macro Format" for each system service indicates the positional dependencies and keyword names of each argument as shown in the following sample:

$SERVICE arga ,argb ,argc ,argd

This format indicates that the macro name of the service is $SERVICE and that it requires four arguments, ordered as shown and with keyword names ARGA, ARGB, ARGC, and ARGD. The argument list for this service must have the format:

```
31 8 7 0

  0                     4

  arga

  argb

  argc

  argd
```

All arguments are longwords. The first longword in the list must always contain, in its low-order byte, the number of arguments in the remainder of the list. The remaining three bytes must be zeros.

Many arguments to system services are optional; these are indicated in the macro formats by brackets. For example, if the second and third arguments of $SERVICE are optional, the macro format would appear as:

$SERVICE arga ,[argb] ,[argc] ,argd

If you omit an optional argument in a system service macro instruction, the macro supplies a default value for the argument.

There are two generic macro forms for coding calls to system services:

$\text{name G}$
$\text{name S}$

2-2
CALLING THE SYSTEM SERVICES
MACRO CODING

The form of the macro to use depends on how the argument list for the system service is constructed:

1. The $name_G form requires you to construct an argument list elsewhere in the program and specify the address of this list as an argument to the system service. (A macro is provided to create an argument list for each system service.) With this form, you can use the same argument list, with modifications if necessary, for more than one invocation of the macro.

2. The $name_S form requires you to supply the arguments to the system service in the macro instruction. The macro generates code to push the argument list onto the call stack during program execution. With this form, you can use registers to contain or to point to arguments so you can write re-entrant programs.

The $name_G macro form generates a CALLG instruction; the $name_S macro form generates a CALLS instruction. The services are called according to the standard procedure calling conventions. System services save all registers except R0 and R1, and restore the saved registers before returning control to the caller.

The following sections describe how to code system service calls using each of these macro forms.

2.1.2 $name_G Form

The $name_G macro form requires a single operand:

$name_G label

label
address of the argument list.

You can use the $name macro to create the argument list. The format of the $name macro is:

label: $name arg1, ..., argn

label
symbolic address of the generated argument list. This is the label given as an argument in the $name_G macro form.

$name
the service macro name.

arg1, ..., argn
arguments to be placed in successive longwords in the argument list.

2.1.2.1 Specifying Arguments with the $name Macro - When you use the $name macro to construct an argument list for a system service, you can specify the arguments in any of three ways:

1. By using keywords to describe the arguments. A keyword must be followed by an equal sign (=) and then by the value of the argument.


2-3
2. In positional order, with omitted arguments indicated by commas in the argument positions. You can omit commas for optional trailing arguments.

3. Using both positional dependence and keyword names (you must list positional arguments first).

For example, $SERVICE may have the format:

$SERVICE arga,[argb] ,[argc] ,argd

Assume, for the purposes of this example, that ARGA and ARGB are arguments that require you to code numeric values and that ARGC and ARGD require you to code addresses.

The two following examples show valid ways of coding a $name macro to construct an argument list for a later call to $SERVICE.

Example 1: Using Keywords

LIST:  $SERVICE ARGB=0,ARGC=0,ARGA=1,ARGD=MYARGD

Example 2: Specifying Arguments in Positional Order

LIST:  $SERVICE 1,...,MYARGD

The argument list generated in both cases is:

LIST:  .LONG  4
       .LONG  1
       .LONG  0
       .LONG  0
       .LONG  MYARGD

Note that all arguments, whether coded in positional order or by keyword, must be expressions that the assembler can evaluate to generate .LONG data directives.

2.1.2.2 Example of $name and $name_G Macro Calls - This example shows how you can code a call to the Read Event Flags ($READEF) system service using an argument list created by $name.

As shown in Chapter 4, the macro format of the $READEF system service is:

$READEF efn ,state

The EFN argument must specify the number of an event flag cluster, and the STATE argument must supply the address of a longword to receive the contents of the cluster.

These arguments might be specified using the $name macro form as follows:

READLIST:  $READEF EFN=1,STATE=TESTFLAG $ ARGUMENT LIST FOR $READEF

This $READEF macro generates the code:

READLIST:  .LONG  2 $ ARGUMENT LIST FOR $READEF
            .LONG  1
            .LONG  TESTFLAG
CALLING THE SYSTEM SERVICES
MACRO CODING

To execute the $READEFS macro now requires only the line:

$READEFS_G READLST

The macro generates the following code to call the Read Event Flags system service:

CALL6 READLST,@$SYS$READEFS

SYS$READEFS is the name of a vector to the entry point of the Read Event Flags system service. The linker automatically resolves the entry point addresses for all system services.

2.1.2.3 **Symbolic Names for Argument List Offsets** - The $Name_G macro form (used with the $Name macro) is especially useful for:

- Coding calls to system services that have long argument lists
- Services that may be called repeatedly during the execution of a single program, with the same, or essentially the same, argument list

When you use this form, you can refer to arguments in the list symbolically. Each argument in an argument list has an offset from the beginning of the list; a symbolic name is defined for the numeric offset of each argument. If you use the symbolic names to refer to the arguments in a list, you do not have to remember the numeric offset (which is based on the position of the argument shown in the macro format). There are two additional advantages to referring to arguments by their symbolic names:

1. Your code is more readable.
2. If an argument list for a system service changes with a later release of a system, the symbols will not change.

The offset names for all system service argument lists are formed by concatenating the service macro name with $_ and the keyword name of the argument, as follows:

\[
\text{name$_{keyword}}
\]

where name is the macro name for the system service and keyword is the keyword argument.

Similarly, the number of arguments required by a particular macro is defined symbolically as:

\[
\text{name$_{NARGS}}
\]

Symbolic names for argument list offsets are defined automatically whenever you use the $Name form of the macro for a particular system service.

For example, the $READEFS macro defines the following values:

<table>
<thead>
<tr>
<th>Symbolic Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>READEFS$_NARGS</td>
<td>Number of arguments in the list (2)</td>
</tr>
<tr>
<td>READEFS$_EPN</td>
<td>Offset of EPN argument (4)</td>
</tr>
<tr>
<td>READEFS$_STATE</td>
<td>Offset of STATE argument (8)</td>
</tr>
</tbody>
</table>
Thus, the $READEF macro can be coded to build an argument list for a $READEF system service call as follows:

    READLST:  $READEF  EFN=1,STATE=TEST1

Later, the program may want to use a different value for the STATE argument in calling the service. The following lines show how this can be accomplished.

    MOVAL  TEST2,READLST+READEF$STATE
    $READEF_G READLST

The MOVAL instruction replaces the address TEST1 in the $READEF argument list with the address TEST2; the $READEF_G macro calls the system service with the modified list.

2.1.2.4 The $nameDEF Macro - You can also define symbolic names for system service argument lists using the $nameDEF macro. This macro does not generate any executable code; it merely defines the symbolic names so they can be used later in the program. For example:

    $QIODEF

This macro defines the symbol QIOS$NARGS and symbolic names for the $QIO argument list offsets.

You may need to use the $nameDEF macro if you code an argument list to a system service without using the $name macro form, or if a program refers to an argument list in a separately assembled module.

2.1.3 The $name_S Form

The format of $name_S macro call is:

    $name_S arg1, ..., argn

The macro generates code to push the arguments on the stack in reverse order. The actual instructions used to place the arguments on the stack are determined as follows:

1. If the system service requires a value for an argument, either a PUSHL instruction or a MOVZWL to -(SP) instruction is generated.

2. If the system service requires an address for an argument, a PUSHAB, PUSHAW, PUSHAL, or PUSHAQ instruction is generated, depending on the context.

The macro then generates a call to the system service in the format:

    CALLS #n,#SYS$name

where n is the number of arguments on the stack.
2.1.3.1 Specifying Arguments with the $name_S Macro - When you use the $name_S macro to construct an argument list for a system service, you can specify arguments in any of three ways:

1. By using keywords to describe the arguments. All keywords must be followed by an equal sign (=) and then by the value of the argument.

2. In positional order, with omitted arguments indicated by commas in the argument positions. You can omit commas for optional trailing arguments.

3. By using both positional dependence and keyword names (positional arguments must be listed first).

For example, $SERVICE might have the format:

$SERVICE arga ,[argb] ,[argc] ,argd

Assume, for the purposes of this example, that ARGA and ARGB are arguments that require you to code numeric values and that ARGC and ARGD require you to code addresses.

The two following examples show valid ways of coding the $name_S macro form to call $SERVICE.

**Example 1: Using Keywords**

MYARGD:

```
.BLKW 1
.
.
$SERVICE_S ARGB=#0,ARGC=0,ARGA=#1,ARGD-MYARGD
```

**Example 2: Specifying Arguments in Positional Order**

MYARGD: .LONG 4

```
.
.
$SERVICE_S $1,,MYARGD
```

The argument list is pushed on the stack as follows:

```
PUSHAW MYARGD
PUSHL $0
PUSHL $0
PUSHL #1
```

Note that all arguments, whether coded positionally or with keywords, must be valid assembler expressions, since they are used as source operands in instructions. Contrast this with the arguments for the $name argument list, which the assembler uses for data-generating directives.
2.1.3.2 Example of $name_S Macro Call - Since a $name_S macro constructs the argument list at execution time, addresses and values can be supplied using register addressing modes. The $READEF macro used in the example of the $name_S form can be coded as follows using the $name_S form:

$READEF $ EFN=1, STATE=(R10)

where R10 contains the address of the longword to receive the status of the flags.

This macro instruction is expanded as follows:

PUSHAL (R10)
PUSHL #1
CALLS $2, @SYS$READEF

2.1.4 Conventions for Coding Arguments to System Services

The arguments must be specified according to the macro assembler rules for operand coding and addressing.

The way to specify a particular argument depends on:

- Whether the system service requires an address or a value as the argument. In Chapter 4, the descriptions of the arguments following a system service macro format always state whether the argument is an address or a value.

- The form of the system service macro being used. The expansions of the $name and $name_S macros in the examples in the preceding sections showed the code generated by each macro form.

If you are in doubt as to whether you have coded a value or an address argument correctly, you can assemble the program with the .LIST MEB directive to check the macro expansion. See the VAX-11 MACRO Language Reference Manual for more details.

Arguments that are optional to system services always have default values, regardless of whether they are value or address arguments. In almost every case, an optional argument defaults to 0.

When an argument is optional, the description of the argument always describes what action the service takes when the default value is used.

Address arguments may be optional when the system service returns information; if the program does not require the information, you can omit the optional argument.
2.1.4.1 Conventions for Coding Character String Arguments - Many system services require ASCII text name strings as arguments or return ASCII text name strings. Character strings are identified to system services by specifying the address of a quadword character string descriptor that has the format:

```
31  16  15  0
   0  length
   address
```

length
is a word specifying the length of the string (in bytes).

address
is a longword containing the address of the string.

When a service returns a character string, you must supply the address of a quadword character string descriptor that describes the length and address of an output buffer into which the string is to be written. Optionally, you can supply the address of a word (16 bits) to receive the actual length of the string returned.

Example of Coding a Character String Descriptor: The Translate Logical Name ($TRNLOG) system service uses character string descriptors for both input and output: it accepts a logical name for input and returns the equivalence name, if any, for the logical name. The following example shows how these descriptors might be coded to translate the logical name CYGNUS.

```
CYGNUSDESC:
  .LONG 20$-10$
  .LONG 10$
  10$:
  .ASCII /CYGNUS/
  20$:
NAMEDESC:
  .LONG 40$-30$
  .LONG 30$
  30$:
  .BLKB 63
  40$:
NAMELENGTH:
  .BLKW 1
  *TRNLG_S LOGNAME=CYGNUSDESC,RSLLEN=NAMELENGTH,-RSLBUF=NAMEDESC
```
The input string for this service call is defined at the label CYGNUSDESC. The output string that is returned from the service will be written into the 63-byte buffer defined in the descriptor at the label NAMEDESC. The actual length of the returned string will be written in the word at the label NAMELENGTH.

When an output buffer is provided for a character string, and the string returned is longer than the buffer, the string returned is truncated, and the service returns a status code indicating that fact. (Status codes returned by system services are discussed in Section 2.1.5.)
CALLING THE SYSTEM SERVICES
MACRO CODING

A Macro to Create Character String Descriptors: Because many system services use character string descriptors, you may want to write a macro to create them. The following example shows such a macro:

```
.MACRO DESCRIPTOR TEXT,?LABEL1,?LABEL2
.LONG LABEL2-LABEL1
.LONG LABEL1
LABEL1: .ASCII /TEXT/
LABEL2:
.ENDM DESCRIPTOR
```

If this macro were used in the example above to create the character string descriptor for the input name CYGNUS, it might be coded as follows:

```
CYGNUSDESC: DESCRIPTOR <CYGNUS>
```

Note that this macro, named DESCRIPTOR, is used in the examples in Chapter 3 whenever a character string descriptor is required for input.

2.1.4.2 Conventions for Coding Numeric Values - Many system services accept numeric values for particular arguments. In some cases, the services check only the low-order portion of the longword argument they are passed. These cases are:

- Indicators. Indicators can only have values of 0 or 1. System services check only the low-order bit of these arguments.
- Event flag numbers. Event flag numbers can have values of 0 through 255. System services check only the low-order byte of these arguments.
- Access modes. Access modes can have values of 0 through 3. System services check only the low-order 2 bits of these arguments.

When you code any of the above types of argument, the high-order portion of the argument should be zeros.

Note that many system services use access modes to protect system resources, and thus employ a special convention for interpreting access mode arguments (keyword ACMODE). You can specify an access mode using a numeric value or a symbolic name. The access modes, their numeric values, and symbolic names are:

<table>
<thead>
<tr>
<th>Access Mode</th>
<th>Numeric Value</th>
<th>Symbolic Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kernel</td>
<td>0</td>
<td>PSLSC_KERNEL</td>
</tr>
<tr>
<td>Executive</td>
<td>1</td>
<td>PSLSC_EXEC</td>
</tr>
<tr>
<td>Supervisor</td>
<td>2</td>
<td>PSLSC_SUPER</td>
</tr>
<tr>
<td>User</td>
<td>3</td>
<td>PSLSC_USER</td>
</tr>
</tbody>
</table>

The symbolic names are defined in the $PSLDEF macro.
When you specify an access mode the actual mode used is determined after the service has compared the specified access mode with the access mode from which the service was called. If the modes are different, the less privileged access mode is always used. Because this operation results in an access mode with a higher numeric value (when the access mode of the caller is different from the specified access mode), the access modes are said to be maximized.

Since much of the code you write will execute in user mode, you can omit the access mode argument. The argument value defaults to 0, and when this value is compared with the current execution mode, the mode with the higher value, 3 for user mode, is used.

2.1.5 Status Codes Returned from System Services

When a system service finishes execution, a numeric status value is always returned in general register RO. Successful completion is indicated by a status code with the low-order bit set. The low-order three bits, taken together, represent the severity of the error. Severity code values are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Warning</td>
</tr>
<tr>
<td>1</td>
<td>Success</td>
</tr>
<tr>
<td>2</td>
<td>Error</td>
</tr>
<tr>
<td>3</td>
<td>Informational</td>
</tr>
<tr>
<td>4</td>
<td>Severe or fatal error</td>
</tr>
<tr>
<td>5-7</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

The remaining bits in the low-order word classify the particular return condition. The high-order word indicates that a system service issued this status code.

Each numeric status code has a unique symbolic name in the format:

SS$_code$

where code is a mnemonic describing the return condition. For example, a successful return is indicated by

SS$_NORMAL$

An example of an error return status code is:

SS$_ACCVIO$

This status code indicates that an access violation occurred because a service could not read an input field or write an output field.

You can obtain the symbolic definitions for status codes at assembly time by coding the system macro SSSDEF. Use the symbolic names for system status codes to check return conditions, because the numeric values for status codes may change with a later release of the system.

2.1.5.1 Information Provided by Status Codes - Status codes returned by system services may provide information, that is, they do not always just indicate whether or not the service completed successfully. SS$_NORMAL$ is the usual status code indicating success, but others are defined. For example, the status code SS$_BUFFEROVF$, 2-11
which is returned when a character string returned by a service is longer than the buffer provided to receive it, is a success code. This status code, however, gives the program additional information.

Warning returns, and some error returns, indicate that the service may have performed some part, but not all, of the requested function.

The possible status codes that each service can return are described with the individual service descriptions in Chapter 4. When you are coding calls to system services, read the descriptions of the return status codes to determine whether you want the program to check for particular return conditions.

2.1.5.2 Testing Return Status Codes - To test for successful completion following a system service call, the program can test the low-order bit of R0 and branch to an error checking routine if this bit is not set, as follows:

\[
\text{BLBC } R0, \text{errlabel} \quad \text{ERROR IF LOW BIT CLEAR}
\]

The error checking routine may check for specific values or for specific severity levels. For example, the following instruction checks for an illegal event flag number error condition:

\[
\text{CMPW } SS\_\text{ILLEFC}, R0 \quad \text{IS EVENT FLAG NUMBER ILLEGAL?}
\]

Note that return status codes are always longword values; however, since the high-order words of all status codes returned by system services are always the same, you need only check the low-order word.

2.1.5.3 System Messages Generated by Status Codes - When you execute a program with the DCL command RUN, the command interpreter uses the contents of R0 to issue a descriptive message if the program completes with a nonsuccessful status.

The following example shows a simple error checking procedure:

\[
\text{READF_\_S EFN=\#64,STATE=\_TEST}
\]

\[
\text{BSBW ERROR}
\]

\[
\text{ERROR: BLBC R0,10\$ \quad \text{CHECK REGISTER 0}}
\]

\[
\text{RSB SUCCESS, RETURN}
\]

\[
\text{10\$: RET \quad \text{EXIT WITH RO STATUS}}
\]

Following a system service call, the BSBW instruction calls the subroutine ERROR. The subroutine checks the low-order bit in register 0 and if the bit is clear, branches to a RET instruction that causes the program to exit with the status of R0 preserved. Otherwise, the subroutine issues an RSB to return to the main program.

If the event flag cluster requested in this call to $READF is not currently available to the process, the program exits and the command interpreter displays the message:

\[
\text{SYSTEM-F-UNASEFC, unassociated event flag cluster}
\]

The keyword UNASEFC in the message corresponds to the status code SS$ UNASEFC.
2.1.5.4 Special Return Conditions - Two process execution modes affect how control is returned to the calling program when an error occurs during the execution of a system service. These modes are:

- Resource wait mode
- System service failure exception mode

If you change the default setting for either of these modes in a program, the program must handle the special return conditions that result. The next two sections discuss considerations for using these modes.

**Resource Wait Mode:** Many system services require certain system resources for execution. These resources include system dynamic memory and process quotas for I/O operations. Normally, when a system service is called and a required resource is not available, the process is placed in a wait state until the resource becomes available. Then, the service completes execution. This mode is called resource wait mode.

However, in a time-critical environment, it may not be practical or desirable for a program to wait: in these cases, you can choose to disable resource wait mode, so that when a required resource is unavailable, control returns immediately to the calling program with an error status code. You can disable (and re-enable) resource wait mode with the Set Resource Wait Mode ($SETRWM) system service.

How a program responds to the unavailability of a resource depends very much on the application and the particular service that is being called. In some instances, the program may be able to continue execution and retry the service call later. In other instances, it may be necessary only to note that the program is being required to wait.

**System Service Failure Exception Mode:** When an error occurs during the execution of a system service, control normally returns to the next instruction in the calling program, which can check the return status code in R0 to determine the success or failure of the service call.

To detect and respond to system service call failures, you can use the condition handling mechanism of VAX/VMS to respond to system service failures. Then, when an error occurs, a software exception condition is generated, and control is passed to a condition handling routine.

This mode is called system service failure exception mode, and can be enabled (and disabled) with the Set System Service Failure Exception Mode ($SETSFM) system service. For example:

```
$SETSF0 $ENRFLG=1
```

This call enables the generation of exceptions when errors or severe errors occur during execution of a system service (exceptions are not generated for warning returns).

If you code a program to execute with this mode enabled, you can code a condition handling routine. Information on condition handlers is provided in Section 3.7, "Condition Handling Services." If no user-specified routine is available when an exception occurs, and the program was run with the DCL command RUN, the default condition handler causes the program to exit and displays descriptive information about the exception condition.
2.2 FORTRAN CODING

If you are a VAX-11 FORTRAN IV-PLUS programmer, you can code calls to system services using either of two FORTRAN language constructs:

- A subroutine CALL statement
- A function reference

The method you choose depends on whether you want the program to check the return status value following the completion of the system service. If you use a function reference, you can have the program check for specific values on return from the service to determine the success or failure of the request.

The use of each of these methods is discussed in this section.


2.2.1 The Format for Calling System Services

You can determine the arguments required by a system service from the service description in Chapter 4. The "High Level Language Format" in each system service description indicates the service name and the positional dependencies of its arguments. For example:

```
SYSSERVICE (arga ,argb ,argc ,argd ,arge)
```

This sample format indicates that the name of the service is $SERVICE, and that its procedure name is SYSSERVICE. SYSSERVICE is the name of a vector to the procedure that executes the service; the entry point addresses for all system services are automatically resolved by the linker.

The format also indicates that $SERVICE requires five arguments. You must code the arguments in parentheses following the procedure name; use commas to separate the arguments.

Many arguments to system services are optional; these are indicated in the format by brackets ([ ]). For example, if the third and fifth arguments of $SERVICE are optional, the format would appear as:

```
SYSSERVICE (arga ,argb ,[argc] ,argd ,[arge])
```

If you omit an optional argument, you must include a comma to indicate the absence of the argument. For example, if the format of $SERVICE is as shown above, and you choose to omit the optional arguments, you could code either of the following:

```
CALL SY$name(arga ,argb ,argd )     !SUBROUTINE CALL
```

or

```
integer_variable=SY$name(arga ,argb ,argd )     !FUNCTION REFERENCE
```

Note that a comma is required to indicate the absence of each optional trailing argument.

When you omit an optional argument, the compiler supplies a default value of 0.

2-14
2.2.1.1 Example of a Subroutine CALL – The following example shows how the Read Event Flags (S$READEF) system service might be called from a FORTRAN program.

The format of the S$READEF system service as shown in Chapter 4 is:

```
SYS$READEF (efn ,state)
```

The EFN argument must specify the number of an event flag and the STATE argument must supply the address of a variable to receive the status of the flags in the cluster.

These arguments might be specified in a subroutine CALL statement as follows:

```
INTEGER*4 TSTFLG       ! RECEIVE STATUS FROM READEF
                 :
                 :
CALL SYS$READEF(XVAL(1),TSTFLG) ! CALL READ EVENT FLAGS
```

This statement requests that the status of the event flags in event flag cluster 0 be returned in the variable TSTFLG.

The use of the XVAL built-in function, and the declaration of TSTFLG as INTEGER*4 are programming considerations for coding arguments to system services in a FORTRAN program. These considerations are discussed in Section 2.2.2, "Conventions for Coding Arguments to System Services".

2.2.1.2 Example of a Function Reference – When you use a function reference, you can assign the return status value from the system service to an INTEGER*4 variable. You must also declare the service name as INTEGER*4 so the function value returned will be in the correct format.

Using the same arguments of the S$READEF system service as shown in the preceding example, a function reference might be coded as follows:

```
INTEGER*4 TSTFLG,SYS$READEF,ICODE ! OUTPUT AND STATUS OF READEF
                 :
                 :
ICODE = SYS$READEF(XVAL(1),TSTFLG) ! READ THE FLAGS
```

Again, the variable TSTFLG is declared to receive the status of flags in the cluster. The system service function SYS$READEF is declared as an INTEGER*4 function (external reference).

For additional examples of function references, see Section 2.2.3, "Status Codes Returned from System Services."
2.2.2 Conventions for Coding Arguments to System Services

Arguments that are expressed as variables or constants must be declared or specified according to the VAX-11 FORTRAN IV-PLUS syntax rules.

The way to specify a particular argument depends on:

- Whether the service requires an address or a value as the argument
- The data type of the argument (if the service requires the address of the argument)

The descriptions of the arguments following the system service format always state whether an address is required. If the description does not say "address," you must provide a value.

The argument descriptions contain terms that may not be familiar to you as a FORTRAN programmer. Table 2-1 lists the terms that are used in Chapter 4 to describe arguments and illustrates how these arguments can be coded in a FORTRAN program.

The following sections provide additional details on value and address arguments.
### Table 2-1

FORTRAN Arguments for System Services

<table>
<thead>
<tr>
<th>Argument Type</th>
<th>Constant</th>
<th>Variable</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>byte value</strong></td>
<td><code>SYS$name(10)</code></td>
<td><code>BYTE ABC - or - LOGICAL*1 ABC</code></td>
<td><code>BYTE ABC - or - LOGICAL*1 ABC</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>SYS$name('VAL(10)')</code></td>
<td><code>SYS$name('VAL(10)+ABC')</code></td>
</tr>
<tr>
<td><strong>byte address</strong></td>
<td><code>SYS$name(10)</code></td>
<td><code>BYTE ABC - or - LOGICAL*1 ABC</code></td>
<td><code>SYS$name(ABC+10)</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>SYS$name(10)</code></td>
<td><code>SYS$name(ABC)</code></td>
</tr>
<tr>
<td><strong>word value</strong></td>
<td><code>SYS$name('VAL(1234)')</code></td>
<td><code>INTEGER*2 DEF - or - LOGICAL*2 DEF</code></td>
<td><code>INTEGER*2 DEF - or - LOGICAL*2 DEF</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>SYS$name('VAL(DEF)')</code></td>
<td><code>SYS$name('VAL(DEF+1234)')</code></td>
</tr>
<tr>
<td><strong>word address</strong></td>
<td><code>SYS$name(1234)</code></td>
<td><code>INTEGER*2 DEF - or - LOGICAL*2 DEF</code></td>
<td><code>INTEGER*2 DEF - or - LOGICAL*2 DEF</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>SYS$name(DEF)</code></td>
<td><code>SYS$name(DEF+1234)</code></td>
</tr>
<tr>
<td><strong>longword value</strong></td>
<td><code>SYS$name('VAL(1234)')</code></td>
<td><code>INTEGER*4 GHI - or - LOGICAL*4 GHI</code></td>
<td><code>INTEGER*4 GHI - or - LOGICAL*4 GHI</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>SYS$name('VAL(GHI)')</code></td>
<td><code>SYS$name('VAL(GHI+1234)')</code></td>
</tr>
<tr>
<td><strong>longword address</strong></td>
<td><code>SYS$name(40000)</code></td>
<td><code>INTEGER*4 GHI - or - LOGICAL*4 GHI</code></td>
<td><code>INTEGER*4 GHI - or - LOGICAL*4 GHI</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>SYS$name(GHI)</code></td>
<td><code>SYS$name(GHI+40000)</code></td>
</tr>
<tr>
<td><strong>quadword (64-bit value)</strong></td>
<td>----</td>
<td><code>INTEGER*4 JKL(2)</code></td>
<td>----</td>
</tr>
<tr>
<td><strong>2-longword array</strong></td>
<td>----</td>
<td><code>SYS$name(JKL)</code></td>
<td>----</td>
</tr>
<tr>
<td><strong>character string descriptor</strong></td>
<td><code>SYSSname('ALPHA')</code></td>
<td><code>CHARACTER*15 NAME</code></td>
<td><code>CHARACTER*15 NAME</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>SYSSname(NAME)</code></td>
<td><code>SYSSname(NAME/'DAT')</code></td>
</tr>
<tr>
<td><strong>entry mask or routine</strong></td>
<td>----</td>
<td><code>EXTERNAL PROGA</code></td>
<td>----</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>SYSSname(PROGA)</code></td>
<td><code>SUBROUTINE PROGA</code></td>
</tr>
</tbody>
</table>

**Note:** For input arguments, you can use constants, variables, or expressions. For output arguments, you must use variables.
2.2.2.1 Value Arguments - All value arguments must be indicated by enclosing the value expression within the built-in function %VAL, in the format:

%VAL(value_expression)

Values can be expressed as constants, variables, or expressions, as in the following examples:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>%VAL(1234)</td>
<td>Constant value</td>
</tr>
<tr>
<td>INTEGER*4 ABC</td>
<td>Declare variable</td>
</tr>
<tr>
<td>%VAL(ABC)</td>
<td>Use current value of variable</td>
</tr>
<tr>
<td>%VAL(ABC+1234)</td>
<td>Use current value of variable plus constant</td>
</tr>
</tbody>
</table>

Some arguments are designated in the service descriptions as:

- Indicators
- Access modes

Indicators are arguments that can have only one of two values, 0 or 1. You can specify these arguments as byte, word, or longword values; however, system services check only the low-order bit of the argument.

Access modes are used by the operating system to provide memory protection; they can have the following values:

<table>
<thead>
<tr>
<th>Access Mode</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kernel</td>
<td>0</td>
</tr>
<tr>
<td>Executive</td>
<td>1</td>
</tr>
<tr>
<td>Supervisor</td>
<td>2</td>
</tr>
<tr>
<td>User</td>
<td>3</td>
</tr>
</tbody>
</table>

The values can be specified as byte, word, or longword values; however, system services check only the low-order 2 bits of these arguments. You can omit the access mode argument when you code a system service call. For more details on how system services interpret this argument, see Section 2.1.4.2, "Conventions for Coding Numeric Values."

2.2.2.2 Address Arguments - System services may require addresses to refer to either input values or output variables. When you code address arguments, you must consider how the argument is used (for input or output) and the data type (that is, the length of the argument) that is required. Table 2-1 summarizes the data types that system services can require and gives examples of valid coding.

2.2.2.3 Input Address Arguments - For input address arguments that refer to byte, word, or longword values, you can supply either constant values, variable names, or expressions in the system service call.
CALLING THE SYSTEM SERVICES
FORTRAN CODING

In all cases, if you supply a variable name for the argument, the variable data type must be equal to or larger than the data type required, as follows:

- If a byte is required, use BYTE, INTEGER*2, or INTEGER*4
- If a word is required, use INTEGER*2 or INTEGER*4
- If a longword is required, use INTEGER*4

If the address refers to a quadword (64-bit) or 2-longword array, you must declare a properly dimensioned array.

When a service requires the "address of an entry mask," or the "address of a routine," you must declare an external procedure. For example:

EXTERNAL PROGA

This statement defines the procedure PROGA for an input argument to a system service.

2.2.2.4 Output Address Arguments - For output address arguments, you must declare a variable to receive the value returned, so that storage is allocated for the output.

When a value is returned, you must declare a variable of the required length to receive the value. For example, the Get Time ($GETTIM) system service returns a quadword binary time value. You can code a call to this service as follows:

INTEGER*4 SYSTIM(2)
;
;
CALL SYS$GETTIM(SYSTIM)

2.2.2.5 Conventions for Coding Character String Arguments - Many system services require ASCII text name strings as input arguments or return ASCII strings. For these arguments, the description of the argument in Chapter 4 refers to a "character string descriptor."

When a system service requires the address of a character string descriptor for an input argument, you can code either a character constant in the system service call or you can provide the name of a variable that has been declared as CHARACTER. The VAX-11 FORTRAN IV-PLUS compiler automatically generates the character string descriptor required for the argument.

When a system service requires the address of a character string descriptor to return a character string, you must provide the name of a variable that has been declared as CHARACTER to receive the string. Optionally, you can supply the name of an INTEGER*2 variable to receive the length of the string returned.
Example of Coding Character String Arguments: The Translate Logical Name ($TRNLOG) system service requires the addresses of character string descriptors for both input and output arguments: it accepts a logical name for input and returns the equivalence name, if any, of the logical name. These arguments might be coded as follows to translate the logical name CYGNUS.

```
CHARACTER*63 CYGNAM
INTEGER*2 CYGLEN

CALL SYS$TRNLOG('CYGNUS',CYGLEN,CYGNAM,...) !TRANSLATE CYGNUS
```

In the above example, the input logical name, CYGNUS, is coded as a character constant in the system service call. When the $TRNLOG system service completes, it places the equivalence name string in the character variable CYGNAM, and places the length (the number of characters in the equivalence name string) in the variable CYGLEN.

2.2.2.6 Default Values for Optional Arguments - Arguments that are optional to system services always default to 0, regardless of whether they are value or address arguments.

When an argument is optional, its description always indicates what action the service takes when the default value is used. Address arguments are often optional when the system service returns information; if the program does not require the information, you can omit the optional argument.

Remember that you must always indicate the absence of an optional argument by entering a comma.

2.2.3 Status Codes Returned from System Services

When you code a system service call using a function call statement, a status code from the system service is returned as an INTEGER*4 function value. The low-order bit of this longword indicates successful or nonsuccessful completion of the service.

The low-order three bits, taken together, represent the severity of the error. Severity code values are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Severity Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Warning</td>
</tr>
<tr>
<td>1</td>
<td>Success</td>
</tr>
<tr>
<td>2</td>
<td>Error</td>
</tr>
<tr>
<td>3</td>
<td>Informational</td>
</tr>
<tr>
<td>4</td>
<td>Severe, or fatal, Error</td>
</tr>
<tr>
<td>5-7</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

The remaining bits classify the particular return condition, and the operating system component that issued the status code.
CALLING THE SYSTEM SERVICES
FORTRAN CODING

Each numeric status code has a symbolic name in the format:

SS$\_code

where code is a mnemonic describing the return condition. For example, a successful return is indicated by:

SS$\_NORMAL

An example of an error status code is:

SS$\_ACCVIO

This status code indicates that a service could not read an input argument or write an output argument.

2.2.3.1 Information Provided by Status Codes - Status codes returned by system services may provide information; that is, they do not always just indicate whether or not the service completed successfully. SS$\_NORMAL is the usual status code indicating success, but others are defined. For example, the status code SS$\_BUFFEROVF, which is returned when a character string returned by a service is longer than the buffer provided to receive it, is a successful code. This status code, however, gives the program more information than that provided by SS$\_NORMAL.

Warning returns, and some error returns, indicate that the service may have performed some part, but not all, of the requested function.

The possible status codes that each service can return are described with the individual service descriptions in Chapter 4. When you are coding calls to system services, read the descriptions of the return status codes to determine whether you want the program to check for particular return conditions.

2.2.3.2 Testing Return Status Codes - When you code a call to a system service using a function reference, you can follow the service call with a logical test on the function value defined for the service call, where TRUE indicates successful completion. For example, a $READEF statement may be coded:

```fortran
INTEGER*4 SYS$READEF,TSTFLG,I

I = SYS$READEF(ZVAL(1),TSTFLG) ! CALL READEF AS FUNCTION
IF (.NOT. I) GOTO 90000 ! ERROR IF FALSE
```

In the above example, the variable I is tested following the call to the $READEF system service. If a nonsuccessful status code is returned, the program branches; otherwise, it continues execution.

These statements may also be combined, for example:

```fortran
INTEGER*4 SYS$READEF,TSTFLG

I = SYS$READEF(ZVAL(1),TSTFLG) ! CALL READEF AS FUNCTION
IF (.NOT. I) GOTO 90000 ! ERROR IF READEF FAILS
```

See COURSE SS-28 for error message.
CALLING THE SYSTEM SERVICES
FORTRAN CODING

You can also code calls to services that check for particular errors following the function reference; or, you may want to provide a GOTO statement (as in the above examples) to branch to a procedure that checks for specific errors.

The following example illustrates a program checking for a particular error return from the $READEF system service:

```
INTEGER*4 SYSSREADEF, TSTFLG, ICODE

ICODE = SYSSREADEF(2, VAL(2), TSTFLG)
IF (ICODE .EQ. SS$_ILLEFC) GOTO 90000
```

The symbolic definitions for system status codes are maintained in the default system library, STARLET.MLB. If your program is going to test for these specific return values, you must create an INCLUDE file to define the symbol names as parameters.

Use these symbolic names whenever you code tests for return status values, since the numeric values may change with a later release of the system.

Appendix A "System Symbolic Definition Macros" describes how to obtain the numeric values for system symbols. For more information on INCLUDE files containing system symbols, see the VAX-11 FORTRAN IV-PLUS User's Guide.

2.2.3.3 Special Return Conditions - Two process execution modes affect how control is returned to the calling program when an error occurs during the execution of a system service. These modes are:

- Resource wait mode
- System service failure exception mode

If you choose to change the default setting for either of these modes, your program must handle the special return conditions that result.

**Resource Wait Mode:** Many system services require certain system resources for execution. These resources include system dynamic memory and process quotas for I/O operations. Normally, when a system service is called and a required resource is not available, the program is placed in a wait state until the resource becomes available. Then, the service completes execution. This mode is called resource wait mode.

However, in a time-critical environment, it may not be practical or desirable for a program to wait; in these cases, you can choose to disable resource wait mode, so that when such a condition occurs, control returns immediately to the calling program with an error status code. You can disable (and re-enable) resource wait mode with the Set Resource Wait Mode ($SETRWM) system service.
How a program responds to the unavailability of a resource depends very much on the application and the particular service that is being called. In some instances, the program may be able to continue execution and retry the service call later. In other instances, it may be necessary only to note that the program is being required to wait.

**System Service Failure Exception Mode:** System service failure exception mode determines whether control is returned to the caller in the normal manner following an error in a system service call, or whether an exception condition is generated. System service failure exception mode is disabled by default; the calling program receives control following an error. It is recommended that FORTRAN programs do not enable system service failure exception mode.
CHAPTER 3
HOW TO USE SYSTEM SERVICES

This chapter presents background and usage information on:

- Event flag services
- AST (Asynchronous System Trap) services
- Logical name services
- Input/output services
- Process control services
- Timer and time conversion services
- Condition handling services
- Memory management services

Whenever possible, coding examples (using VAX-11 MACRO) are given to familiarize you with the system services and their arguments. The examples do not show complete programming sequences; rather, they show only the code and/or arguments that are pertinent to a particular discussion.

In some of the more complex examples, explanatory text is keyed to the example using a special numeric symbol, for example 4.

The examples are coded using VAX-11 MACRO. If you are a FORTRAN programmer, see Figure 3-1 for an explanation of how to interpret the MACRO examples.
Notes:

1. In the MACRO example, a routine name and entry mask show the beginning of executable code in a routine or subroutine; in FORTRAN, the routine and its entry mask are defined by the SUBROUTINE statement.

2. The MACRO examples define input character string arguments with a DESCRIPTOR macro. This is not necessary in FORTRAN; you can code an input character string directly in the system service call.

3. These three MACRO directives declare a 63-byte buffer for an output character string. In FORTRAN, the CHARACTER*63 declaration is all that is necessary.

4. The MACRO directive .BLKW reserves a word for an output value. This is equivalent to the FORTRAN INTEGER*2 declaration.

5. A MACRO programmer calls a system service by a macro name, which does not have the "SYS" portion of the procedure name. A macro name for a system service call has an _S or _G suffix. Note the following differences between MACRO and FORTRAN in the manner of coding arguments:
   a. MACRO arguments are not placed in parentheses.
   b. MACRO arguments in the examples are specified with a keyword name preceding the actual argument. These names correspond to the names of the arguments shown in lowercase in the system service formats in Chapter 4. FORTRAN arguments must be coded in the positional order shown in Chapter 4.
   c. No indication is given when an optional argument is omitted in a MACRO argument list that uses keywords; you must code a comma when you omit an optional argument in FORTRAN.
   d. The MACRO programmer uses a number sign character (#) to indicate a literal value for an argument. This is equivalent to the $VAL function in FORTRAN.

6. The MACRO examples show a check for an error return from a system service with the BSBW instruction; this is equivalent to a FALSE logical test following a function reference in FORTRAN.

Figure 3-1 FORTRAN Interpretation of MACRO Examples
MACRO Example

CYGDES: DESCRIPTOR <CYGNUS>
NAMDES: .LONG 20$-10$
          .LONG 10$
10$:  .BLKB 63
20$:  
NAMLEN: .BLKW 16

ORION::

:WORD 0

*TRNLOG,S LOGNAM=CYGDES,RSLLEN=NAMLEN,RSLBUF=NAMDES,-
   TABLE=1
BSBW   ERROR

:END

FORTRAN Equivalent

SUBROUTINE ORION
   !PROCEDURE ORION
   
   CHARACTER*63 NAMDES
   !OUTPUT BUFFER DESCRIPTOR
   INTEGER*2 NAMLEN
   !WORD TO RECEIVE LENGTH
   INTEGER*4 SYS$TRNLOG
   !DEFINE SYSTEM SERVICE FUNCTION

   ICODE = SYS$TRNLOG('CYGNUS',NAMLEN,NAMDES,%VAL(1),)

   IF (.NOT. ICODE) GOTO 90000
      !BRANCH IF ERROR

   END

Figure 3-1 (Cont.) FORTRAN Interpretation of MACRO Examples
3.1 EVENT FLAG SERVICES

Event flags are status posting bits maintained by VAX/VMS for general programming use. Some system services set an event flag to indicate the completion or the occurrence of an event; the calling program can test the flag. For example, the Queue I/O Request ($QIO) system service sets an event flag when the requested input or output operation completes.

Programs can use event flags to perform a variety of signaling functions:

- Setting or clearing specific flags
- Testing the current status of flags
- Placing the current process in a wait state pending the setting of a specific flag or a group of flags

Moreover, event flags can be used in common by more than one process, as long as the cooperating processes are in the same group. Thus, if you have developed an application that requires the concurrent execution of several processes, you can use event flags to establish communication among them and to synchronize their activity.

3.1.1 Event Flag Numbers and Event Flag Clusters

Each event flag has a unique decimal number; event flag arguments in system service calls refer to these numbers. For example, if you specify event flag 1 when you code a $QIO system service, then event flag number 1 is set when the I/O operation completes.

To allow manipulation of groups of event flags, the flags are ordered in clusters, with 32 flags in each cluster, numbered from right to left, corresponding to bits 0 through 31 in a longword. The clusters are also numbered. The range of event flag numbers encompasses the flags in all clusters: event flag 0 is the first flag in cluster 0, event flag 32 is the first flag in cluster 1, and so on.

There are two types of cluster:

1. A local event flag cluster can only be used internally by a single process. Local clusters are automatically available to each process.

2. A common event flag cluster can be shared by cooperating processes in the same group. Before a process can refer to a common event flag cluster, it must explicitly "associate" with the cluster. Association is described in Section 3.1.4, "Common Event Flag Clusters."

The ranges of event flag numbers and the clusters to which they belong are summarized in Table 3-1.

3.1.1.1 Specifying Event Flag and Event Flag Cluster Numbers — The same system services manipulate flags in both local and common event flag clusters. Since the event flag number implies the cluster number, you do not have to specify the cluster number when you code a system service call that refers to an event flag.
HOW TO USE SYSTEM SERVICES
EVENT FLAG SERVICES

Table 3-1
Summary of Event Flag and Cluster Numbers

<table>
<thead>
<tr>
<th>Cluster Number</th>
<th>Event Flag Numbers</th>
<th>Description</th>
<th>Restriction</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1</td>
<td>0-31</td>
<td>Process-local event flag clusters for general use</td>
<td>Event flags 24 through 31 are reserved for system use</td>
</tr>
<tr>
<td>2 3</td>
<td>64-95 96-127</td>
<td>Assignable common event flag cluster</td>
<td>Must be associated before use</td>
</tr>
</tbody>
</table>

When a system service requires an event flag cluster number as an argument, you need only specify the number of any event flag that is in the cluster. Thus, to read the event flags in cluster 1, you could specify any number in the range 32 through 63.

3.1.2 Examples of Event Flag Services

Local event flags are most commonly used with other system services. For example, with the Set Timer ($SETIMR) system service you can request that an event flag be set at a specific time of day, or after a specific interval of time has passed. If you want to place a process in a wait state for a specified period of time, you could code an event flag number for the $SETIMR service, and then use the Wait for Single Event Flag ($WAITFR) system service, as follows:

```
TIME: .RKEO 1
      $WILL CONTAIN TIME INTERVAL TO WAIT
      
      $SETIMR_S EFN=#33, DAYTIM=TIME $SET THE TIMER
      $WAITFR_S EFN=#33 $WAIT UNTIL TIMER EXPIRES
```

In this example, the DAYTIM argument refers to a time value. Details on how to obtain a time value in the proper format for input to this service are contained in Section 3.6, "Timer and Time Conversion Services."

3.1.2.1 Event Flag Waits - Three system services place the process in a wait state pending the setting of an event flag:

- The Wait for Single Event Flag ($WAITFR) system service places the process in a wait state until a single flag has been set.
- The Wait for Logical OR of Event Flags ($WFLOR) system service places the process in a wait state until any one of a specified group of event flags has been set.
- The Wait for Logical AND of Event Flags ($WFLAND) places the process in a wait state until all of a specified group of flags have been set.
HOW TO USE SYSTEM SERVICES
EVENT FLAG SERVICES

Another system service that accepts an event flag number as an argument is the Queue I/O Request ($QIO) system service. Figure 3-2 shows a program that issues two $QIO system service calls, and uses the $WFLAND system service to wait until both I/O operations complete before it continues execution.

```
$QIO.S EFN=#1,...
$QIO.S EFN=#2,...
$WFLAND S EFN=#1, MASK=#B0110

\textbf{Notes:}
\begin{enumerate}
  \item The event flag argument is specified in each $QIO request. Both of these event flags are in cluster 0.
  \item After both I/O requests are successfully queued, the program calls the Wait for Logical AND of Event Flags ($WFLAND) system service to wait until the I/O operations are completed. In this service call, the EFN argument corresponds to a cluster number: the cluster that contains event flag 1, that is, cluster 0. The MASK argument specifies which flags in the cluster are to be waited for: flags 1 and 2.
\end{enumerate}

\textbf{Figure 3-2 Using Local Event Flags}

3.1.3 Setting and Clearing Event Flags

The $SETMR and $QIO system services clear the event flag specified in the system service call before they queue the timer or I/O request. This ensures the integrity of the event flag with respect to the process. If you are using event flags in local clusters for other purposes, take care to verify the state of a flag before you use it.

The Set Event Flag ($$SETEF) and Clear Event Flag ($$CLREF) system services set and clear specific event flags. For example, the following system service call clears event flag 32:

```
$CLREF.S EFN=#32
```

The $$SETEF and $$CLREF services return successful status codes that indicate whether the flag specified was set or clear when the service was called. The caller can thus verify the previous state of the flag, if necessary. The codes returned are SS$_WASSET$ and SS$_WASCLR$.

Event flags in common event flag clusters are all initially clear when the cluster is created. The next section describes the creation of common event flag clusters.
3.1.4 Common Event Flag Clusters

Before any processes can use event flags in a common event flag cluster, the cluster must be created: the Associate Common Event Flag Cluster ($ASCEFC) system service creates a common event flag cluster. Once a cluster has been created, other processes in the same group can call $ASCEFC to establish their association with the cluster, so they can access flags in it.

When a common event flag cluster is created, it must be identified by a 1- to 15-character name string. All processes that associate with the cluster must use the same name to refer to the cluster; the $ASCEFC system service establishes the correspondence between the cluster name and the cluster number that a process assigns to it.

The following example shows how a process might create a common event flag cluster named COMMON CLUSTER and assign it a cluster number of 2:

```
CLUSTER:
  DESCRIPTOR <COMMON CLUSTER> CLUSTER NAME
  ...
  $ASCEFC-S EFN='65;NAME=COMMON CLUSTER CREATE CLUSTER 2
```

Subsequently, other processes in the same group may associate with this cluster. Those processes must use the same character string name to refer to the cluster; but the cluster numbers they assign do not have to be the same.

Common event flag clusters are either temporary or permanent. The PERM argument to the $ASCEFC system service defines whether the cluster is temporary or permanent.

Temporary clusters:

- Require an element of the creating process's quota for timer queue entries (TQELM quota).
- Are deleted when all processes associated with the cluster have disassociated. Disassociation can be performed explicitly, with the Disassociate Common Event Flag Cluster ($DACFEC) system service, or implicitly, when the image exits.

Permanent clusters:

- Require the creating process to have the PRMCEB user privilege.
- Continue to exist until they are explicitly marked for deletion with the Delete Common Event Flag Cluster ($DLCFEC) system service.

If cooperating processes that are going to use a common event flag cluster all have the requisite privilege or quota to create a cluster, the first process to call the $ASCEFC system service creates the cluster.
3.1.5 Disassociating and Deleting Common Event Flag Clusters

When a process no longer needs access to a common event flag cluster, it issues the Disassociate Common Event Flag Cluster ($DACEFC) system service. When all processes associated with a temporary cluster have issued a $DACEFC system service, the system deletes the cluster. If a process does not explicitly disassociate itself from a cluster, the system performs an implicit disassociation when the image that called $ASCEFC exits.

Permanent clusters, however, must be explicitly marked for deletion with the Delete Common Event Flag Cluster ($DLCEFC) system service. After the cluster has been marked for deletion, it is not deleted until all processes associated with it have been disassociated.

3.1.6 Example of Using a Common Event Flag Cluster

Figure 3-3 shows an example of four cooperating processes that share a common event flag cluster. The processes named ORION, CYGNUS, LYRA, and PEGASUS are in the same group.

Notes on Figure 3-3:

1. Assume for this example that ORION is the first process to issue the $ASCEFC system service, and therefore is the creator of the cluster. Since this is a newly created cluster, all event flags in it are 0.

2. The argument NAME in the $ASCEFC system service call is a pointer to the descriptor CNAME for the name to be assigned to the cluster: in this example, the cluster is named COMMON CLUSTER. This service call associates the name COMMON CLUSTER with cluster 2, containing event flags 64 through 95. Cooperating processes CYGNUS, LYRA, and PEGASUS must use the same character string name to refer to this cluster.

3. The continuation of process ORION depends on work done by processes CYGNUS, LYRA, and PEGASUS. The Wait For Logical AND of Event Flags ($WPLANL) system service call specifies a mask indicating the event flags that must be set before process ORION can continue. The mask in this example, "XE is the hexadecimal equivalent of binary 1110: it indicates that the second, third, and fourth flags in the cluster must be set.

4. Process CYGNUS executes, associates with the cluster, sets event flag 65, and disassociates.

5. Process LYRA associates with the cluster, but instead of referring to it as cluster 2, it refers to it as cluster 3 (with event flags in the range 96 through 127). Thus, when process LYRA sets flag 99, it is setting the fourth bit in COMMON CLUSTER.

6. Process PEGASUS associates with the cluster, waits for an event flag set by process LYRA, and sets an event flag itself.

7. When all three event flags are set, process ORION continues execution and calls the $DACEFC system service. Since ORION did not specify the PERM argument when it created the cluster, COMMON CLUSTER is deleted.
HOW TO USE SYSTEM SERVICES
EVENT FLAG SERVICES

Process ORION

CNAME: DESCRIPTOR <COMMON CLUSTER> ; CLUSTER NAME

1) $ASCEFC_S EFN=#64, NAME=CNAME ; CREATE COMMON CLUSTER
   BSBW    ERROR ; CHECK FOR ERROR

2) $WFLAND_S EFN=#64, MASK=^XE ; WAIT FOR FLAGS 1, 2, 3
   BSBW    ERROR ; CHECK FOR ERROR

3) $DACEFC_S EFN=#64 ; DISASSOCIATE CLUSTER

Process CYGNUS

ORION_FLAGS: DESCRIPTOR <COMMON CLUSTER> ; CLUSTER NAME

4) $ASCEFC_S EFN=#64, NAME=ORION_FLAGS
   BSBW    ERROR ; CHECK FOR ERROR
   $SETEF_S EFN=#65 ; SET EVENT FLAG 1
   BSBW    ERROR ; CHECK FOR ERROR
   $DACEFC_S EFN=#64 ; DISASSOCIATE

Process LYRA

SHARE: DESCRIPTOR <COMMON CLUSTER> ; CLUSTER NAME

5) $ASCEFC_S EFN=#96, NAME=SHARE ; ASSOCIATE WITH CLUSTER 3
   BSBW    ERROR ; CHECK FOR ERROR
   $SETEF_S EFN=#99 ; SET FLAG 3
   BSBW    ERROR ; CHECK FOR ERROR
   $DACEFC_S EFN=#96 ; DISASSOCIATE

Process PEGASUS

CLUSTER: DESCRIPTOR <COMMON CLUSTER> ; CLUSTER NAME

6) $ASCEFC_S EFN=#64, NAME=CLUSTER ; ASSOCIATE WITH CLUSTER
   BSBW    ERROR ; CHECK FOR ERROR
   $WAITFR_S EFN=#65 ; WAIT FOR FLAG 1
   BSBW    ERROR ; CHECK FOR ERROR

   $SETEF_S EFN=#66 ; SET FLAG 2
   BSBW    ERROR ; CHECK FOR ERROR
   $DACEFC_S EFN=#64 ; DISASSOCIATE

Figure 3-3 Example of a Common Event Flag Cluster
3.2 AST (ASYNCHRONOUS SYSTEM TRAP) SERVICES

Some system services allow a process to request that it be interrupted when a particular event occurs. Since the interrupt occurs asynchronously (out of sequence) with respect to the process's execution, the interrupt mechanism is called an asynchronous system trap (AST). The trap provides a transfer of control to a user-specified routine that handles the event.

The system services that use the AST mechanism accept as an argument the address of an AST service routine, that is, a routine to be given control when the event occurs.

These services are:

- Queue I/O Request ($QIO)
- Set Timer ($SETIMR)
- Set Power Recovery AST ($SSETPRA)
- Update Section File on Disk ($UPDSEC)

For example, if you code a Set Timer ($SETIMR) system service, you can specify the address of a routine to be executed when a time interval expires, or at a particular time of day. The service sets the timer and returns; the program image continues executing. When the requested timer event occurs, the system "delivers" an AST by interrupting the process and calling the specified routine.

The following sections describe in more detail how ASTs work and how to use them.

3.2.1 Example of an AST

Figure 3-4 shows a typical program that calls the $SETIMR system service with a request for an AST when a timer event occurs.

Notes on Figure 3-4:

1. The call to the $SETIMR system service requests an AST at 12:00 noon.

   The DAYTIM argument refers to the quadword NOON, which must contain the time in system time format. For details on how this is done, see Section 3.6, "Timer and Time Conversion Services." The ASTADR argument refers to TIMEAST, the address of the AST service routine.

   When the call to the system service completes, the process continues execution.

2. The timer expires at 12:00 and notifies the system. The system interrupts execution of the process and gives control to the AST service routine.

3. The user routine TIMEAST handles the interrupt. When the AST routine completes, it issues a RET instruction to return control to the program. The program resumes execution at the point at which it was interrupted.
3.2.2 Access Modes for AST Execution

Each request for an AST is qualified by the access mode from which the AST is requested. Thus, if an image executing in user mode requests notification of an event by means of an AST, the AST service routine executes in user mode.

Since the ASTs you use will almost always execute in user mode, you do not need to be concerned with access modes. However, you should be aware of some system considerations for AST delivery. These considerations are described in Section 3.2.6, "AST Delivery."

3.2.3 ASTs and Process Wait States

A process that is in a wait state can be interrupted for the delivery of an AST and the execution of an AST service routine. When the AST service routine completes execution, the process is returned to the wait state, if the condition that caused the wait is still in effect.

The following wait states may be interrupted:

- Event flag waits
- Hibernation
- Resource waits and page faults

3.2.3.1 Event Flag Waits - If a process is waiting for an event flag and is interrupted by an AST, the wait state is restored following execution of the AST service routine. If the flag is set during the execution of the AST service routine (for example, by completion of an I/O operation) then the process continues execution when the AST service routine completes.

Event flags are described in detail in Section 3.1, "Event Flag Services."
3.2.3.2 Hibernation - A process can place itself in a wait state with the Hibernate ($HIBER) system service. This wait state can be interrupted for the delivery of an AST. When the AST service routine completes execution, the process continues hibernation. The process can, however, "wake" itself in the AST service routine or be awakened by another process or as the result of a timer scheduled wakeup request. Then, it continues execution when the AST service routine completes.

Process suspension is another form of wait; however, a suspended process cannot be interrupted by an AST. Process hibernation and suspension are described in Section 3.5, "Process Control Services."

3.2.3.3 Resource Waits and Page Faults - When a process is executing an image, the system can place the process in a wait state until a required resource becomes available, or until a page in its virtual address space is paged into memory. These waits, which are generally transparent to the process, can also be interrupted for the delivery of an AST.

3.2.4 How ASTs Are Declared

Most ASTs occur as the result of the completion of an asynchronous event initiated by a system service, for example, a $QIO or $SEETIMR request, when the process requests notification by means of an AST.

There is also a system service that creates ASTs: the Declare AST ($DCLAST) system service. With this service, a process can declare an AST only for the same or for a less privileged access mode.

You may find occasional use for the $DCLAST system service in your programming applications; you may also find the $DCLAST service useful when you want to test an AST service routine.

3.2.5 The AST Service Routine

An AST service routine must be a separate routine. The system calls the AST with a CALLG instruction; the routine must return using a RET instruction. If the service routine modifies any registers other than R0 or R1, it must set the appropriate bits in the entry mask so that the contents of those registers are saved.

Since it is impossible to know when the AST service routine will begin executing, you must take care, when you code the AST service routine, that the service routine does not modify any data or instructions used by the main procedure.
HOW TO USE SYSTEM SERVICES
AST (ASYNCHRONOUS SYSTEM TRAP) SERVICES

On entry to the AST service routine, the Argument Pointer register (AP) points to an argument list that has the format:

<table>
<thead>
<tr>
<th>31</th>
<th>8</th>
<th>7</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td></td>
<td>5</td>
</tr>
</tbody>
</table>

- **AST parameter**
- R0
- R1
- PC
- PSL

The registers R0 and R1, the PC, and PSL in this list are those that were saved when the process was interrupted by delivery of the AST.

The AST parameter is an argument passed to the AST service routine so that it can identify the event that caused the AST. When you code a call to a system service requesting an AST, or when you code a $DCLAST system service, you can supply a value for the AST parameter. If you do not specify a value, it defaults to 0.

Figure 3-5 illustrates an AST service routine. In this example, the ASTs are created by the $DCLAST system service: the ASTs are delivered to the process immediately, so that the service routine is called following each $DCLAST system service call.

### 3.2.6 AST Delivery

When an AST occurs, the system may not be able to deliver the interrupt to the process immediately. An AST cannot be delivered if any of the following conditions exist:

1. An AST service routine is currently executing at the same or at a more privileged access mode.

   ASTs are implicitly disabled when an AST service routine executes, so that one AST routine cannot be interrupted by another AST routine declared for the same access mode. It can, however, be interrupted by an AST declared for a more privileged access mode.

2. AST delivery is explicitly disabled for the access mode.

   A process can disable the delivery of AST interrupts with the Set AST Enable ($SETAST) system service. This service may be useful when a program is executing a sequence of instructions that should not be interrupted for the execution of an AST routine.
3. The process is executing at an access mode more privileged than that for which the AST is declared.

For example, if a user mode AST is declared as the result of a system service, but the program is currently executing at a higher access mode (because of another system service call, for example), the AST is not delivered until the program is once again executing in user mode.

If an AST cannot be delivered when the interrupt occurs, the AST is queued until the condition(s) disabling delivery are removed. Queued ASTs are ordered by the access mode from which they were declared, with those declared from more privileged access modes at the front of the queue. If more than one AST is queued for an access mode, the ASTs are delivered in the order in which they are queued.

PEGASUS: .WORD 0
    .
    .
    .
    $DCLAST_S ASTADR=ASTRTN,ASTPRM=#1
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    RET

ASTRTN: .WORD 0
    .
    .
    .
    .
    CMPL $1,4(AP)
    BEQ 10$
    CMPL $2,4(AP)
    BEQ 20$
    10$: RET
    20$: RET

END PEGASUS

Notes:

1. The program PEGASUS calls the Declare AST system service twice to queue ASTs. Both ASTs specify the AST service routine, ASTRTN. However, a different parameter is passed for each call.

2. The first action that this AST routine takes is to check the AST parameter, so that it can determine if the AST being delivered is the first or second one declared. The value of the AST parameter determines the flow of execution.

Figure 3-5 The AST Service Routine
3.3 LOGICAL NAME SERVICES

The VAX/VMS logical name services provide a technique for manipulating and substituting character string names. Logical names are commonly used to specify devices or files for input or output operations. You can code programs with logical, or symbolic, names to refer to physical devices or files, and then establish an equivalence, or actual, name by issuing the ASSIGN command from the command stream before program execution. When the program executes, a reference to the logical name results in the substitution of the equivalence name.

This section describes how to use system services to establish logical names for general application purposes. For specific details on logical name usage for I/O system services, see Section 3.4, "Input/Output Services" in this manual, and the discussion of logical names in the VAX/VMS Command Language User's Guide.

3.3.1 Logical Names and Equivalence Names

Logical name and equivalence name strings can have a maximum of 63 characters. You can establish logical name and equivalence name pairs:

1. At the command level, with the ALLOCATE, ASSIGN, DEFINE, or MOUNT commands

2. In a program, with the Create Logical Name ($CRELOG) and Create Mailbox and Assign Channel ($CREMBX) system services

For example, you could use the symbolic name TERMINAL to refer to an output terminal in a program. For a particular run of the program, you could use the ASSIGN command to establish the equivalence name TTA2:

To perform an assignment in a program, you must provide character string descriptors for the name strings and use the $CRELOG system service as shown in the following example. In either case, the result is the same: the logical name TERMINAL is equated to the physical device name TTA2:

`TERMINAL: DESCRIPTOR <TERMINAL> ;DESCRIPTOR FOR LOGICAL NAME
TTNAME: DESCRIPTOR <TTA2> ;DESCRIPTOR FOR EQUIVALENCE

$CRELOG $TBFLG=2,LOGNAM=TERMINAL,EDLNAM=TTNAME`

The TBFLG argument in this example indicates the logical name table number, in this case, the process logical name table. Logical name tables and logical name table numbers are discussed in the following sections.
3.3.2 Logical Name Tables

Logical name and equivalence name pairs are maintained in three logical name tables:

- Process
- Group
- System

A process logical name table contains names used exclusively by the process. A process logical name table exists for each process in the system. Some entries in the process logical name table are made by system programs executing at more privileged access modes; these entries are qualified by the access mode from which the entry was made. For example, logical names created at the command level are supervisor mode entries.

The group logical name table contains names that cooperating processes in the same group can use. The GRPNAM privilege is required to place a name in the group logical name table.

The system logical name table contains names that all processes in the system can access. This table includes the default names for all system-assigned logical names. The SYSNAM privilege is required to place a name in the system logical name table.

Figure 3-6 illustrates some sample logical name table entries.

Notes on Figure 3-6:

1. This process logical name table equates the logical name TERMINAL to the specific terminal TTA2:. INFILE and OUTFILE are equated to disk file specifications: these logical names were created from supervisor mode.

2. The group logical name table shows entries qualified by group numbers; only processes that have the indicated group number can access these entries.

3. In Group 100, the logical name TERMINAL is equated to the terminal TTA1:. Individual processes in Group 100 that want to refer to the logical name TERMINAL do not have to individually assign it an equivalence name.

4. Group 200 has entries for logical names MAILBOX and DISPLAY. Other processes in group 200 can use these logical names for input or output operations.

5. In Group 300, the logical name TERMINAL is equated to the physical device name TTA3:. Note that there are two entries for TERMINAL in the group logical name table. These are discrete entries, since they are qualified by the number of the group to which they belong.

6. The system logical name table contains the default physical device names for all processes in the system. SYS$LIBRARY and SYS$SYSTEM provide logical names for all users to refer to the device(s) containing system files.
Logical Name Table for Process A (Group Number = 200)

<table>
<thead>
<tr>
<th>Logical Name</th>
<th>Equivalence Name</th>
<th>Access Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>TERMINAL</td>
<td>TTA2:</td>
<td>User</td>
</tr>
<tr>
<td>INFILE</td>
<td>DM1:[HIGGINS]TEST.DAT Supervisor</td>
<td></td>
</tr>
<tr>
<td>OUTFILE</td>
<td>DM1:[HIGGINS]TEST.OUT Supervisor</td>
<td></td>
</tr>
</tbody>
</table>

Group Logical Name Table

<table>
<thead>
<tr>
<th>Logical Name</th>
<th>Equivalence Name</th>
<th>Group Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 TERMINAL</td>
<td>TTA1:</td>
<td>100</td>
</tr>
<tr>
<td>4 MAILBOX</td>
<td>MB3:</td>
<td>200</td>
</tr>
<tr>
<td>DISPLAY</td>
<td>TERMINAL</td>
<td>200</td>
</tr>
<tr>
<td>5 TERMINAL</td>
<td>TTA3:</td>
<td>300</td>
</tr>
</tbody>
</table>

System Logical Name Table

<table>
<thead>
<tr>
<th>Logical Name</th>
<th>Equivalence Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYS$LIBRARY</td>
<td>DBA0:[SYSLIB]</td>
</tr>
<tr>
<td>SYS$SYSTEM</td>
<td>DBA0:[SYSTEM]</td>
</tr>
</tbody>
</table>

Figure 3-6 Logical Name Table Entries

3.3.2.1 Logical Name Table Numbers - Each logical name table has a number associated with it. To place an entry in a logical name table, specify a logical name table number with the TBLFLG argument to the $CRELOG system service. The logical name table numbers are as follows:

<table>
<thead>
<tr>
<th>Table</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process</td>
<td>2</td>
</tr>
<tr>
<td>Group</td>
<td>1</td>
</tr>
<tr>
<td>System</td>
<td>0</td>
</tr>
</tbody>
</table>

The TBLFLG argument defaults to a value of 0, that is, the system logical name table.

3.3.2.2 Duplication of Logical Names - The process logical name table can contain entries for the same logical name at different access modes. The group logical name table can contain entries for the same logical name, as long as the group numbers are different.
HOW TO USE SYSTEM SERVICES
LOGICAL NAME SERVICES

In all other cases, there can be only one entry for a particular logical name in a logical name table. For example, if the logical name TERMINAL is equated to TTA2: in the process table as shown in the figure, and the process subsequently equates the logical name TERMINAL to TTA3: the equivalence of TERMINAL to TTA2: is replaced by the new equivalence name. The successful return status code SS$SUPERSEDE indicates that a new entry replaced an old one.

Any number of logical names can have the same equivalence name.

3.3.3 Logical Name Translation

When you refer to a logical name for a physical device in an I/O service, the service performs logical name translation automatically. In many cases, a program must perform the logical name translation to obtain the equivalence name for a logical name. The Translate Logical Name ($TRNLOG) system service searches the logical name tables for a specified logical name and returns the equivalence name.

By default, the process, group, and system tables are all searched, in that order, and the first match found is returned. Thus if identical logical names exist in the process and group tables, the process table entry is found first, and the group table is not searched. When the process logical name table is searched, the entries are searched in order of access mode, with user mode entries matched first, supervisor second, and so on.

The following example shows a call to the $TRNLOG system service to translate the logical name TERMINAL.

```
TLOGDESC: DESCRIPTOR <TERMINAL>
  TLOGDESC:
   LONG 20$-10$
   LONG 10$
10$: BLKB 64
20$: TLEN:
   BLKW 1
             #RECEIVE EQLNAM LENGTH HERE

$TRNLOG_S LOGNAME=TLOGDESC,RSLEN=TLEN,RSLBUF=TEQLDESC
```

If the logical name table entries are as shown in Figure 3-6, this call to the $TRNLOG system service results in the translation of the logical name TERMINAL. The equivalence name string TTA2: is placed in the output buffer described by TEQLDESC. The length of the equivalence name string is written into the word at TLEN.

Note that the call to $TRNLOG might be coded as follows:

```
$TRNLOG_S LOGNAME=TLOGDESC,RSLEN=TEQLDESC,RSLBUF=TEQLDESC
```

Then, the output equivalence name string length is written into the first word of the character string descriptor. This descriptor can then be used as input to another system service.

3.3.3.1 Bypassing Logical Name Tables - To disable the search of a particular logical name table, you can code the optional argument DSBMSK to the $TRNLOG system service. This argument is a mask that disables the search of one or more logical name tables. The format of the mask is described in the discussion of the $TRNLOG system service in Chapter 4.
3.3.3.2 Logical Name and Equivalence Name Format Conventions - The operating system uses special conventions for logical name/equivalence name assignments and translation. These conventions are generally transparent to user programs; however, you should be aware of the programming considerations involved.

If a logical name string is preceded with an underscore character (_), $TRNLOG will not translate the logical name. Instead, it returns the status code SS$ NOTRAN, strips the underscore from the logical name string, then writes the string into the result buffer. This convention permits bypassing logical name translation in I/O services when physical device name strings are specified.

At login, the system creates default logical name table entries for process permanent files. The equivalence names for these entries, for example, SY$INPUT and SY$OUTPUT, are preceded with a 4-byte header that contains the following:

<table>
<thead>
<tr>
<th>Byte(s)</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>^X1B (Escape character)</td>
</tr>
<tr>
<td>1</td>
<td>^X00</td>
</tr>
<tr>
<td>2-3</td>
<td>RMS Internal File Identifier (IFI)</td>
</tr>
</tbody>
</table>

This header is followed by the equivalence name string. If any of your program applications must translate system-assigned logical names, the program must be prepared to check for the existence of this header and then to use only the desired part of the equivalence string.

For an example of how to do this, see Figure 3-8 in Section 3.4.7, "Complete Terminal I/O Example."

3.3.4 Recursive Translation

When a translate request is made for a logical name string, the $TRNLOG system service searches the logical name tables only once. If you structure a logical name table or tables such that logical name equivalencies are several levels deep (that is, that an equivalence name is entered in the table as a logical name with another equivalence name, and so on), you may require recursive logical name translation. Note that Figure 3-6 illustrates recursive entries: the logical name DISPLAY is equated to the string TERMINAL in the group table, and the name TERMINAL is equated to the device name string TTA2: in the process table. The $TRNLOG system service must be used twice to complete the translation of the logical name DISPLAY.

You can code a program loop so that the output string from the $TRNLOG service is reused as the input string, and check for the status code SS$ NOTRAN following the call to the service. SS$ NOTRAN indicates that no logical name was found, and that the input string has been written into the output buffer.

3.3.5 Deleting Logical Names

The Delete Logical Name ($DELLLOG) system service deletes entries from a logical name table. When you code a call to the $DELLLOG system service, you can specify a single logical name to delete, or you can
specify that you want to delete all logical names from a particular table. For example, the following call deletes all names from the process logical name table that were entered in the table from user mode:

$DELOG_S TBLFLG=$2

Logical names that were placed in the process logical name table from an image running in user mode are automatically deleted at image exit. Entries made from the command stream are placed in the table by the command interpreter; these are supervisor mode entries, and are not deleted at image exit.
3.4 INPUT/OUTPUT SERVICES

There are two methods you can use to perform input/output operations under VAX/VMS:

- VAX-11 Record Management Services (RMS)
- I/O system services

VAX-11 RMS provides a set of macros for general-purpose, device-independent functions, such as data storage, retrieval, and modification.

The I/O system services permit you to use the I/O resources of the operating system directly in a device-dependent manner. I/O services also provide some specialized functions not available in RMS. Using I/O services requires more knowledge on your part, but can result in more efficient input/output operations.

This section provides general information on how to use the I/O services, including:

- Assigning channels
- Queuing I/O requests
- Allocating devices
- Using mailboxes

Examples are provided to show how to use the I/O services for simple functions, for example, terminal input and output operations. If you plan to write device-dependent I/O routines, see the VAX/VMS I/O User's Guide.

3.4.1 Assigning Channels

Before any input or output operation can be done to a physical device, a channel must be assigned to the device to provide a path between the process and the device. The Assign I/O Channel ($ASSIGN) system service establishes this path.

When you code a call to the $ASSIGN service, you must supply the name of the device, which may be a physical device name or a logical name, and the address of a word to receive the channel number. The service returns a channel number, and you use this channel number when you code an input or an output request.

For example, the following lines assign an I/O channel to the device TTA2. The channel number is returned in the word at TTPAP.

TTNAME: DESCRIPTOR <$TA2>
TTCHAN: .BLKW 1

$ASSIGN_S DEVMAX=TTNAME,CHAN=TTCHAN

To assign a channel to the current default input or output device, you must first translate the logical name SYS$INPUT or SYS$OUTPUT with the Translate Logical Name ($TLMLOG) system service. Then, specify the equivalence name returned as the DEVMAX argument to the $ASSIGN system
service. This technique requires you to interpret header information preceding the equivalence name string for these devices. For an example of this technique, see Figure 3-8 later in this section.

For more details on how $ASSIGN and other I/O services handle logical names, see Section 3.4.10 "Logical Names and Physical Device Names."

3.4.2 Queuing I/O Requests

All input and output operations in VAX/VMS are initiated with the Queue I/O Request ($QIO) system service. $QIO queues the request and returns; while the operating system processes the request, the program that issued the request can continue execution.

Required arguments to the $QIO service include the channel number assigned to the device on which the I/O is to be done, and a function code (expressed symbolically) that indicates the specific operation to be performed. Depending on the function code, one to six additional parameters may be required.

For example, the IO$WRITEVBLK and IO$READVBLK function codes are device-independent codes used to read and write single records or virtual blocks. These function codes are suitable for simple terminal I/O. They require parameters indicating the address of an input or output buffer and the buffer length. A call to $QIO to write a line to a terminal might appear as:

```
$QIO_S  CHAN=TTCHAN,FUNC=$IO$WRITEVBLK, -
P1=BUFADDR,F2=BUFLEN
```

Function codes are defined for all supported device types, and most of the codes are device dependent, that is, they perform functions that are specific to a particular device. The $IODEF macro defines symbolic names for these function codes. The codes are summarized in Appendix A, "System Symbolic Definition Macros;" for details on all function codes and an explanation of the parameters required by each, see the VAX/VMS I/O User's Guide.

3.4.3 Synchronizing I/O Completion

The $QIO system service returns control to the calling program as soon as the I/O request is queued; the status code returned in R0 indicates whether or not the request was queued successfully. To ensure proper synchronization of the I/O operation with respect to the program, the program must:

1. Test for the completion of the I/O operation
2. Test whether the I/O operation itself completed successfully

Optional arguments to the $QIO service provide techniques for synchronizing I/O completion. There are three methods you can use to test for the completion of an I/O request:

- Specify the number of an event flag to be set when the I/O completes
- Specify the address of an AST routine to be executed when the I/O completes
HOW TO USE SYSTEM SERVICES
INPUT/OUTPUT SERVICES

• Specify the address of an I/O status block in which the system can place the return status when the I/O completes.

Examples of using these three techniques are shown in Figure 3-7.

Example 1: Event Flags

```
$QIO_S EFN=$1,...  #ISSUE 1ST I/O REQUEST
BSBW ERROR
$QIO_S EFN=$2,...  #ISSUE 2ND I/O REQUEST
BSBW ERROR
WLAND_S EFN=$0, MASK=$B110  #WAIT TIL BOTH DONE
```

Notes on Example 1:

1. When you code an event flag number as an argument, $QIO clears the event flag when it queues the I/O request. When the I/O completes, the flag is set.

2. In this example, the program issues two I/O requests. A different event flag is specified for each request.

3. The Wait for Logical AND of Event Flags ($WLAND) system service places the process in a wait state until both I/O operations are complete. The EFN argument indicates that the event flags are both in cluster 0; the MASK argument indicates the flags that are to be waited for.

Example 2: An AST Routine

```
$QIO_S ...ASTADR=TTAST,ASTPRM=#1,...  ;I/O WITH AST
BSBW ERROR  ;QUEUED SUCCESSFULLY?

TTAST: .WORD 0  ;AST SERVICE ROUTINE ENTRY MASK
.
.
RET  ;END OF SERVICE ROUTINE
```

Notes on Example 2:

1. When you code the ASTADR argument to the $QIO system service, the system interrupts the process when the I/O completes and passes control to the specified AST service routine.

2. The $QIO system service call specifies the address of the AST routine, TTAST, and a parameter to pass as an argument to the AST service routine. When $QIO returns control, the process continues execution.

3. When the I/O completes, the routine TTAST is called, and it responds to the I/O completion.

When this routine is finished executing, control returns to the process at the point at which it was interrupted.

Figure 3-7 Synchronizing I/O Completion
HOW TO USE SYSTEM SERVICES
INPUT/OUTPUT SERVICES

Example 3: The I/O Status Block

Example 3: The I/O Status Block

TTIOSB: .BLKQ 1

ISSUE I/O REQUEST
BSBW ERROR

$QIO clears the quadword when it queues the I/O request. When the request is successfully queued, the program continues execution.

TSTW TTIOSB
BEQ 10%
CMPW TTIOSB,$SS$NORMAL
BNEQIO_ERR

Figure 3-7 (Cont.) Synchronizing I/O Completion

3.4.4 I/O Completion Status

When an I/O operation completes, the system posts the completion status in the I/O status block, if one is specified. The completion status indicates whether or not the operation actually completed successfully, the number of bytes that were transferred, and additional device-dependent return information.

The format of the information written in the IOSB is:

<table>
<thead>
<tr>
<th>31</th>
<th>16</th>
<th>15</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>count</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>status</td>
</tr>
<tr>
<td></td>
<td>device-dependent information</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
HOW TO USE SYSTEM SERVICES
INPUT/OUTPUT SERVICES

The first word contains a system status code indicating the success or failure of the operation. The status codes used are the same as for all returns from system services; for example, SS$NORMAL indicates successful completion.

The second word contains the number of bytes actually transferred in the I/O operation.

The second longword contains device-dependent return information.

To ensure successful I/O completion and the integrity of data transfers, the IOSB should be checked following I/O requests, particularly for device-dependent I/O functions. For complete details on how to use the I/O status block, see the VAX/VMS I/O User's Guide.

3.4.5 Simplified Forms of the $QIO Macro

The $QIOW macro combines the functions of the $QIO and the Wait for Single Event Flag (SWAITFR) system services. $QIOW has the same arguments as the $QIO macro. It queues the I/O request, and then places the program in a wait state until the I/O is complete.

The $INPUT and $OUTPUT macros are a subset of the $QIOW macro: they use only the function codes to read and write virtual blocks or records (IO$READVBLK and IO$WRITEVBLK, respectively). These macros provide an efficient and easy way to specify I/O for terminals, mailboxes, line printers, and interprocess network transfers.

When you code a $INPUT or $OUTPUT macro, you must specify the channel on which the I/O is to be performed and the length and address of the input or output buffer. Optionally you can specify an event flag to be set when the I/O is complete and the address of an I/O status block. For example:

$INPUT CHAN=TTCHAN, LENGTH=INLEN, BUFFER=INBUF, EFN=#1, IOSB=TTIOSB

or

$OUTPUT CHAN=TTCHAN, LENGTH=OUTLEN, BUFFER=OUTBUF, EFN=#2, IOSB=TTIOSB

3.4.6 Deassigning I/O Channels

When a process no longer needs access to an I/O device, it should release the channel assigned to the device by issuing the Deassign I/O Channel ($DASSIGN) system service. For example:

$DASSIGN_S CHAN=TTCHAN

This service call releases the terminal channel assignment acquired in the $ASSIGN example shown earlier. The system automatically deassigns channels for a process when the image that assigned the channel exits.
3.4.7 Complete Terminal I/O Example

Figure 3-8 shows a complete sequence of input and output operations using the $INPUT and $OUTPUT macros to read and write lines to the current default SYS$INPUT device. Note that if the program containing these lines is executed interactively, the input/output is to the current terminal.

```
TTNAME: DESCRIPTOR <SYS$INPUT>  
TTCHAN: .BLK 1
TTIOSB: .BLK 1  
TTIOLEN: .BLK 1
OUTLEN: .BLK 1
INBUF: .BLK 80
DEVDESC: .LONG 63
NADDR: .LONG NAME
NAME: .BLK 63

4 $TRANLOG S LOGNAM=TTNAME,RSLLEN=NLEN,RSLBUF=DEVDESC
CMPB NAME,$'X18
BNEQ 10% 
SUBL $4,NLEN
ADDL $4,NADDR

10$: 5 $ASSIGN S DEVNAM=DEVDESC,CHAN=TTCHAN
BSBW ERROR

6 $INPUT CHAN=TTCHAN,LENGTH=#80,BUFFER=INBUF,IOSB=TTIOSB 
BSBW ERROR
7 CMPW TTIOSB,$'SS$_NORMAL
BNEQ IO_ERR
8 MOVZWL TTIIOLEN,OUTLEN

9 $OUTPUT CHAN=TTCHAN,LENGTH=OUTLEN,BUFFER=INBUF,IOSB=TTIOSB 
BSBW ERROR
CMPW TTIOSB,$'SS$_NORMAL
BNEQ IO_ERR
10 $DIASSGN S CHAN=TTCHAN
BSBW ERROR
```

Figure 3-8  Example of Terminal Input and Output

Notes on Figure 3-8:

1. TTNAME is a character string descriptor for the logical device SYS$INPUT and TTCHAN is a word to receive the channel number assigned to it.

2. The IOSB for the I/O operations is structured so that the program can easily check for the completion status (in the first word) and the length of the input string returned (in the second word).
The string will be read into the buffer INBUF; the longword OUTLEN will contain the length of the string for the output operation.

The Translate Logical Name ($TRNLOG) system service translates the logical name SYS$INPUT. On return from $TRNLOG, the equivalence name is checked for a 4-byte header beginning with an escape character. (This header is present in all process permanent files; see Section 3.3.3.2, "Logical Name and Equivalence Name Format Conventions.")

If this header is present, the program modifies the descriptor for the device name returned, so it can be used as input to $ASSIGN.

$ASSIGN assigns a channel and writes the channel number at TTCHAN.

If the $ASSIGN service completes successfully, the $INPUT macro reads a line from the terminal, and requests that the completion status be posted in the I/O status block defined at TTIOSB.

The process waits until the I/O is complete, then checks the first word in the I/O status block for a successful return. If not successful, the program takes an error path.

Next, the length of the string read is moved into the longword at OUTLEN. This is necessary because the $OUTPUT macro requires a longword argument, and the length field of the I/O status block is only a word long. The $OUTPUT macro writes the line just read to the terminal.

The program performs error checks: first, it ensures that the $OUTPUT macro successfully queued the I/O request; then, when the request is completed, it ensures that the I/O was successful.

When all I/O operations on the channel are finished, the channel is deassigned.

3.4.8 Canceling I/O Requests

If a process must cancel an I/O request that has been queued but not yet completed, it can issue the Cancel I/O On Channel ($CANCEL) system service. All pending I/O requests issued by the process on that channel are canceled.

For example, the $CANCEL system service can be called as follows:

$CANCEL...S CHAN=TTCHAN

In this example, the $CANCEL system service initiates the cancellation of all pending I/O requests to the channel whose number is located at TTCHAN.

The $CANCEL system service returns after initiating the cancellation of the I/O requests. If the call to $QIO specified an event flag, AST service routine, or I/O status block, the system sets the flag, delivers the AST, or posts the I/O status block as appropriate when the cancellation is actually completed.
3.4.9 Device Allocation

Many I/O devices are shareable; that is, more than one process may access the device at a time. Each process, by issuing a $ASSIGN service, is given a channel to the device for I/O operations.

In some cases, a process may need exclusive use of a device so that data is not affected by other processes. To reserve a device for exclusive use you must allocate it.

Device allocation is normally accomplished from the command stream, with the ALLOCATE command. A process can also allocate a device by calling the Allocate Device ($ALLOC) system service. When a device has been allocated by a process, only the process that allocated the device and any subprocesses it creates can assign channels to the device.

When you code the $ALLOC system service, you must provide a device name. The device name specified can be:

- A physical device name, for example, the tape drive MTB3:
- A logical name, for example, TAPE
- A generic device name, for example, MT:

If you specify a physical device name, $ALLOC attempts to allocate the specified device.

If you specify a logical name, $ALLOC translates the logical name and attempts to allocate the physical device name equated to the logical name.

If you specify a generic device name -- that is, if you specify a device type, but do not specify a controller and/or unit number -- $ALLOC attempts to allocate any device available of the specified type. More information on the allocation of devices by generic names is provided in Section 3.4.10.1.

When you specify logical names or generic device names, you must provide fields for the $ALLOC system service to return the name and the length of the physical device that is actually allocated, so you can provide this name as input to the $ASSIGN system service.

Figure 3-9 illustrates the allocation of a tape device specified by the logical name TAPE.

Notes on Figure 3-9:

1. The $ALLOC system service call requests allocation of a device corresponding to the logical name TAPE, defined by the character string descriptor LOGDEV. The argument DEVDESC refers to the buffer provided to receive the physical device name of the device actually allocated, and its length. $ALLOC translates the logical name TAPE and returns the equivalence name string into the buffer at DEVDESC. It writes the length of the string in the first word of DEVDESC.

2. The $ASSIGN command uses the character string returned by the $ALLOC system service as the input device name argument, and requests that the channel number be written into TAPECHAN.
HOW TO USE SYSTEM SERVICES
INPUT/OUTPUT SERVICES

LOGDEV: DESCRIPTOR <TAPE>
DEVDESC:
   .LONG 20$-10$
   .LONG 10$
10$:
20$:
  BLKB 64
  TAPECHAN:
    .BLKW 1

1 $ALLOC_S DEVNAM=LOGDEV,PHYLEN=DEVDESC,PHYBUF=DEVDESC
   BSBW ERROR
2 $ASSIGN_S DEVNAM=DEVDESC,CHAN=TAPECHAN ;ASSIGN CHANNEL
   BSBW ERROR ;CONTINUE WITH I/O

3 $DEASSGN_S CHAN=TAPECHAN
3 $DEALLOCATE TAPE

Figure 3-9 Device Allocation and Channel Assignment

3 When I/O operations are completed, the $DEASSGN system service
deassigns the channel and the $DEALLOC system service
deallocates the device. The channel must be deassigned before the device can be deallocated.

3.4.9.1 Implicit Allocation - Devices that cannot be shared by more
than one process, for example, terminals and line printers, do not
have to be explicitly allocated. Since they are nonshareable, they
are implicitly allocated by the $ASSIGN system service when $ASSIGN is
called to assign a channel to the device.

3.4.9.2 Deallocation - When the program is finished using an
allocated device, it should release the device with the Deallocate
Device ($DEALLOC) system service, to make it available for other
processes as in this example:

   $DEALLOC_S DEVNAM=DEVDESC

The system automatically deallocates devices allocated by an image at
image exit.

3.4.10 Logical Names and Physical Device Names

When a device name is specified as input to an I/O system service, it
can be a physical device name or a logical name. When an underscore
character (_) precedes a device name string, it indicates that the
string is a physical device name string. For example:

   TTNAMED: DESCRIPTOR <TTB3>
HOW TO USE SYSTEM SERVICES
INPUT/OUTPUT SERVICES

Any string that does not begin with an underscore is considered a logical name, even though it may be a physical device name. The $ASSIGN, $DASSGN, $ALLOC, and $DALLOC system services call the Translate Logical Name ($TRNLOG) system service to search the logical name tables. The $TRNLOG service searches the process, group, and system tables, in that order, and if it locates an entry is found for the specified logical name, the I/O request is performed for the device specified in the equivalence name string. The search is not recursive.

If $TRNLOG does not locate an entry for the logical name, the I/O service treats the name that is specified as a physical device name. When you code the name of an actual physical device in a call to one of these services, code the underscore character to bypass the logical name translation.

When the $ALLOC system service returns the device name of the physical device that has been allocated, the device name string returned is prefixed with an underscore character. When this name is used for the subsequent $ASSIGN system service, the $ASSIGN service does not attempt to translate the device name.

If you use logical names in I/O service calls, you must be sure to establish a valid device name equivalence before program execution. You can do this by issuing an ASSIGN command from the command stream. Or, the program can establish the equivalence name before the I/O service call with the Create Logical Name ($CRELOG) system service.

For details on how to create and use logical names, see Section 3.3, "Logical Name Services."

3.4.10.1 Device Name Defaults - If, after logical name translation, a device name string in an I/O system service call does not fully specify the device name (that is, device, controller, and unit), the service either provides default values for nonspecified fields, or provides values based on device availability.

The following rules apply:

1. The $ASSIGN, $DASSGN, and $DALLOC system services apply default values as shown in Table 3-2.

2. The $ALLOC system service treats the device name as a generic device name and attempts to find a device that satisfies the components of the device name that are specified, as shown in Table 3-2.
Table 3-2
Default Device Names for I/O Services

<table>
<thead>
<tr>
<th>Final Device Name Specification</th>
<th>Device Name Defaults for $ASSIGN, $DASSIGN, and $DALLOC</th>
<th>Generic Device Names Used by $ALLOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>DD:</td>
<td>DDAO: (unit 0 on controller A)</td>
<td>DDon: (any available device of the specified type)</td>
</tr>
<tr>
<td>DDC:</td>
<td>DDCO: (unit 0 on controller specified)</td>
<td>DDCn: (any available unit on the specified controller)</td>
</tr>
<tr>
<td>DDN:</td>
<td>DDAN: (unit specified on controller A)</td>
<td>DDAn: (device of specified type and unit on any available controller)</td>
</tr>
<tr>
<td>DDAN:</td>
<td>DDAN:</td>
<td>DDAN:</td>
</tr>
</tbody>
</table>

**Key:**

DD: is the device type specified
C: is the controller specified
n: is any controller
N: is the unit number specified
n: is any unit number

3.4.11 Obtaining Information About Physical Devices

In cases where a generic (that is, nonspecific) device name is used in an I/O service, the program may need to find out what device has actually been used. The Get I/O Channel Information ($GETCHN) system service provides specific information about the physical device to which a channel has been assigned. The Get I/O Device Information ($GETDEV) system service returns information about a device that is identified by its device name. The information returned includes the unit number of the device, as well as additional device characteristics.

When you code the $GETCHN or $GETDEV service, you must provide the address of a buffer or buffers into which the system writes the information. The format of the buffer, and additional details about these services are given in Chapter 4. Details on the device-specific information these services return is given in the VAX/VMS I/O User's Guide.

3.4.12 Formatting Output Strings

When you are preparing output strings for a program, you may need to insert variable information into a string prior to output, or you may need to convert a numeric value to an ASCII string. The Formatted ASCII Output ($FAO) system service performs these functions.
Input to the $FAO service consists of:

1. A control string that contains the fixed text portion of the output and formatting directives. The directives indicate the position within the string where substitutions are to be made, and describe the data type and length of the input values that are to be substituted or converted.

2. An output buffer to contain the string after conversions and substitutions have been made.

3. An optional argument indicating a word to receive the final length of the formatted output string.

4. Parameters that provide arguments for the directive.

Figure 3-10 shows a call to the $FAO system service to format an output string for a $OUTPUT macro. Accompanying notes briefly discuss the input and output requirements of FAO. Complete details on how to use FAO, with additional examples, are provided in the description of the $FAO system service in Chapter 4.

1 $FAO_STR: DESCRIPTOR <FILE !AS DOES NOT EXIST> $FAO CONTROL STRING
2 $FAO_DESC: LONG $FAO_LEN-$FAO_BUF; DESCRIPTOR FOR FAO OUTPUT
   $BLK=$FAO_BUF; ADDRESS OF BUFFER
   $OUTPUT BUFFER
3 $FAO_LEN: LONG 0; RECEIVE LENGTH OF FAO OUTPUT STRING
4 $FAO_S CTRSTR=$FAO_STR,OUTLEN=$FAO_LEN,OUTBUF=$FAO_DESC,-
   L=$FILE_SPEC; PARAMETER FOR FAO
   $BSW=$ERROR
5 $OUTPUT ...,BUFFER=$FAO_BUF,LENGTH=$FAO_LEN
   $BSW=$ERROR

Figure 3-10 Example of Using Formatted ASCII Output Program

Notes on Figure 3-10:

1. $FAO_STR provides the FAO control string. !AS is an example of an FAO directive; it requires an input parameter that specifies the address of a character string descriptor. When FAO is called to format this control string, !AS will be substituted with the string whose address is specified.

2. $FAO_DESC is a character string descriptor for the output buffer; $FAO will write the string into the buffer, and will write the length of the final formatted string in the low-order word of $FAO_LEN. (A longword is reserved so that it can be used for an input argument to the $OUTPUT macro.)

3. $FILE_SPEC is a character string descriptor defining an input string for the FAO directive !AS.
HOW TO USE SYSTEM SERVICES
INPUT/OUTPUT SERVICES

4 The call to $FAO specifies the control string, the output buffer and length fields, and the parameter PL, which is the address of the string descriptor for the string to be substituted.

5 When $FAO completes successfully, $OUTPUT writes the output string:

FILE DMA1:MYFILE.DAT DOES NOT EXIST

3.4.13 Mailboxes

Mailboxes are virtual devices that can be used for communication between processes. Actual data transfer is accomplished by using RMS or I/O services. When a mailbox is created, a channel is assigned to it for use by the creating process. Other processes can then assign channels to the mailbox using the $ASSIGN system service.

The Create Mailbox and Assign Channel ($CREMBX) system service creates the mailbox. The $CREMBX system service identifies a mailbox by a user-specified logical name and assigns it an equivalence name. The equivalence name is a physical device name in the format MBn: where n is a unit number.

When another process assigns a channel to the mailbox with the $ASSIGN system service, it can identify the mailbox by its logical name. $ASSIGN automatically translates the logical name. The process can obtain the MBn: name by translating the logical name (with the $TRNLOG system service), or it can call the Get I/O Channel Information ($GETCHN) system service to obtain the unit number and the physical device name.

Mailboxes are either temporary or permanent; user privileges are required to create either type. $CREMBX enters the logical name and equivalence name for a temporary mailbox in the group logical name table of the process that created it. The system deletes a temporary mailbox when no more channels are assigned to it.

The $CREMBX system service enters the logical name and equivalence name for a permanent mailbox in the system logical name table.

Permanent mailboxes continue to exist until they are specifically marked for deletion with the Delete Mailbox ($DELMBX) system service.

Figure 3-11 shows an example of processes communicating by means of a mailbox. The accompanying notes explain some of the arguments that the $CREMBX system service requires.
HOW TO USE SYSTEM SERVICES
INPUT/OUTPUT SERVICES

Process ORION

MBLOGINAM: DESCRIPITOR <GROUP100_MAILBOX> \ MAILBOX LOGICAL NAME
MBUFFER: \ INPUT BUFFER FOR MAILBOX READS
        .BLKB 128 \ BUFFER OF 128 BYTES
MBUFLEN: .LONG MBUFLEN-MBUFFER \ BUFFER LENGTH
MBXCHAN: .BLKW 1 \ MAILBOX CHANNEL NUMBER
MBXIOSB: .BLKW 1 \ I/O SB FIRST WORD (STATUS)
MBLEN: \ BLKW 1 \ I/O SB 2ND WORD (LENGTH)
        .BLKL 1 \ REMAINDER OF I/O SB
OUTLEN: \ BLKL 1 \ LONGWORD TO GET LENGTH

ORION: \ WORD "M<R2,R3,R4> \ ENTRY MASK
        *CREMBX_S PRMFLG=#0,CHAN=MBXCHAN,MAXMSG=MBUFLEN-1
        BUFQUO=#3B4,PROMSK=#$X0000,LOGNAM=MBLOGINAM
        BSBW ERROR
        *QIO_S CHAN=MBXCHAN,FUNC=#IO$,READVBBLK,IOSB=MBXIOSB,-2
        ASTRADR=MBXAST,P1=MBUFFER,P2=MBUFLEN
        BSBW ERROR

        RET

MBXAST: \ WORD 0 \ FAST ROUTINE ENTRY MASK
        CMPW MBXIOSB,#15$=NORMAL \ I/O SUCCESSFUL?
        BNEQ ASTERR \ BRANCH IF NOT
        MOVZWL MBLEN,OUTLEN \ MAKE LENGTH A LONGWORD
        *OUTPUT ...,BUFFER=MBUFFER,LENGTH=OUTLEN,...
        BSBW ERROR

        RET

Process CYGNUS

MAILBOX: DESCRIPITOR <GROUP100_MAILBOX> \ MAILBOX LOGICAL NAME
MAILCHAN: \ MAILBOX CHANNEL NUMBER
        .BLKW 1
OUTBUF: \ BLKB 128 \ BUFFER FOR OUTPUT MSG DATA
OUTLEN: \ BLKL 1 \ WILL CONTAIN LENGTH OF MSG

CYGNUS: \ WORD "M<R2,R3,R4> \ ENTRY MASK
        *ASSIGN_S DEVNAM=MAILBOX,CHAN=MAILCHAN \ ASSIGN CHANNEL
        BSBW ERROR
        *OUTPUT CHAN=MAILCHAN,BUFFER=OUTBUF,LENGTH=OUTLEN,...
        BSBW ERROR

        RET

Figure 3-11 Mailbox Creation and I/O
HOW TO USE SYSTEM SERVICES
INPUT/OUTPUT SERVICES

Notes on Figure 3-11:

1. Process ORION creates the mailbox and receives the channel number at MBXCHAN.

This PRMFLG argument indicates that the mailbox is a temporary mailbox. The logical name is entered in the group logical name table.

The MAXMSG argument limits the size of messages that the mailbox can receive. Note that the size indicated in this example is the same size as the buffer (MBUFFER) provided for the $QIO request. A buffer for mailbox I/O must be at least as large as the size specified in the MAXMSG argument.

When a process creates a temporary mailbox, the amount of system memory that is allocated for buffering messages is subtracted from the process's buffer quota. Use the BUFQUO argument to specify how much of the process quota you want to be used for mailbox message buffering.

Mailboxes are protected devices. By specifying a protection mask with the PROMSK argument, you can restrict access to the mailbox. (In this example, all bits in the mask are clear, indicating unlimited read and write access.)

2. After creating the mailbox, Process ORION issues a $QIO system service, requesting notification of the completion of I/O (that is, the reception of a message) by means of an AST interrupt (the AST service routine is at MBXAST). The process can continue executing.

3. When a message is sent to the mailbox, the AST is delivered, and ORION responds to the message. ORION gets the length of the message from the first word of the I/O status block at MBXIOSB and places it in the longword OUTLEN so it can pass the length to $OUTPUT.

4. Process CYGNUS assigns a channel to the mailbox, specifying the logical name the process ORION gave the mailbox. The $OUTPUT form of the $QIO system service writes a message from the output buffer provided at OUTBUP.
3.4.13.1 System Mailboxes - The system uses mailboxes for communication among system processes. All system mailbox messages contain, in the first word of the message, a constant that identifies the sender of the message. These constants have symbolic names (defined in the $MSGDEF macro) in the format:

```
MSG$_sender
```

The remainder of the message contains variable information, depending on the system component that is sending the message.

The format of the variable information for each message type is documented with the system function that uses the mailbox.

3.4.13.2 Mailboxes for Process Termination Messages - When a process creates another process, it can specify the unit number of a mailbox as an argument to the Create Process ($CREPRC) system service. When the created process is deleted, the system sends a message to the specified termination mailbox. An example of how to create and use a termination mailbox is provided in Section 3.5.7.2, "Termination Mailboxes."

3.4.13.3 Mailboxes for System Processes - There are a group of I/O services that are used internally by system processes to communicate various kinds of information. These services are:

- Send Message to Accounting Manager ($SNDACC)
- Send Message to Operator ($SNDOPR)
- Send Message to Symbiont Manager ($SNDSMB)

Details on the formats of the messages and the information they provide are given in the individual discussions of these services in Chapter 4.
3.5 PROCESS CONTROL SERVICES

A process is the primary execution agent in VAX/VMS. When you log into the system, the system creates a process for the execution of program images. When you issue the DCL command RUN, you can request the RUN command to create another process to execute an image.

You can also code a program that creates another process to execute a particular image.

Process control services provide techniques for controlling a process or group of processes.

Included in this section are discussions of:

- Subprocesses and detached processes
- The execution context of a process
- Process creation
- Interprocess control and communication
- Process hibernation and suspension
- Image exit and exit handlers
- Process deletion and termination messages

3.5.1 Subprocesses and Detached Processes

A process is either a subprocess or a detached process. A subprocess receives a portion of its creator's resource quotas, and must terminate before the creator. A detached process is fully independent; for example, the process the system creates for you when you log in is a detached process.

The Create Process ($CREPRC) system service creates both subprocesses and detached processes. The ability to create subprocesses is controlled by the PRCLM quota. The ability to create detached processes is controlled by the DETACH privilege.

3.5.2 The Execution Context of a Process

The execution context of a process defines a process to the system. It includes:

- The image that the process is executing
- The input and output streams for the image executing in a process
- Disk and directory defaults for the process
- System resource quotas and user privileges available to a process

When the system creates a detached process as the result of a login, it uses the system authorization file to determine the process's execution context.
HOW TO USE SYSTEM SERVICES
PROCESS CONTROL SERVICES

For example, when you log into the system:

- The process created for you executes the image known as login.
- The terminal you are using is established as the input, output, and error stream for images that the process executes.
- Your disk and directory defaults are taken from the user authorization file.
- The resource quotas and privileges you have been granted by the system manager are associated with the created process.

When you code the $CREPRC system service to create a process, you define the context by specifying arguments to the service.

3.5.3 Process Creation

The following sections show examples of process creation and describe how the arguments you code to the $CREPRC system service define the context of the process.

3.5.3.1 Defining an Image for a Subprocess to Execute - When you code the $CREPRC system service, use the IMAGE argument to provide the process with the name of a program image to execute. For example, the following lines create a subprocess to execute the program image in the file named LIBRA.EXE.

```
PROGNAME: DESCRIPTOR <LIBRA>  ;IMAGE TO EXECUTE

$CREPRC_S IMAGE=PROGNAME  ;CREATE PROCESS TO EXECUTE LIBRA
```

In this example, only a file name is specified; the service uses current disk and directory defaults, performs logical name translation, uses the default file type of EXE, and locates the most recent version of the image file. When the subprocess completes execution of the image, the subprocess is deleted. Process deletion is described in Section 3.5.7.

3.5.3.2 Input, Output, and Error Devices for Subprocesses - When you code the $CREPRC system service you can provide equivalence names for the logical names SYS$INPUT, SYS$OUTPUT, and SYS$ERROR. These logical name/ equivalence name pairs are placed in the process logical name table for the created process.
Figure 3-12 shows an example of defining input, output, and error devices for a subprocess. The notes indicate how these devices are used.

```
INSTREAM: DESCRIPTOR <SUB_MAIL_BOX>
OUTSTREAM: DESCRIPTOR <COMPUTE.OUT>
PRONAME: DESCRIPTOR <COMPUTE.EXE>

$CREPRC_S IMAGE=PRONAME,INPUT=INSTREAM, OUTPUT=OUTSTREAM,ERROR=OUTSTREAM
```

Notes:

1. The INPUT argument equates the equivalence name SUB_MAIL_BOX to the logical name SYS$INPUT. This logical name may represent a mailbox that the calling process previously created with the Create Mailbox and Assign Channel ($CREMBX) system service. Any input the subprocess reads from the logical device SYS$INPUT will be read from the mailbox.

2. The OUTPUT argument equates the equivalence name COMPUTE.OUT to the logical name SYS$OUTPUT. All messages the program writes to the logical device SYS$OUTPUT will be written to this file.

3. The ERROR argument equates the equivalence name COMPUTE.OUT to the logical name SYS$ERROR. All system-generated error messages will be written into this file. Since this is the same file as that used for program output, the file effectively contains a complete record of all output produced during the execution of the program image.

Figure 3-12 Defining Input and Output Streams for a Subprocess

The $CREPRC system service does not provide default equivalence names for these logical names; if none are specified, entries in the group or system logical name tables, if any, may provide equivalences. If, while the subprocess executes, it reads or writes to one of these logical devices and no equivalence name exists, an error condition results.

You can code a program that creates a subprocess to share the logical input, output, or error devices of the creating process. The following steps are required:

- Use the Translate Logical Name ($TRNLOG) system service to obtain the current equivalence name for the logical name.

- Check whether the equivalence name returned contains system header information (a 4-byte field beginning with an escape character); if the logical name table entry was created by the command interpreter, it will contain this header. If there is a header, adjust the length of the string returned and the address of the string returned by modifying these fields in the character string descriptor of the resultant name string.
HOW TO USE SYSTEM SERVICES
PROCESS CONTROL SERVICES

* Specify the address of this descriptor when you code the INPUT, OUTPUT, or ERROR arguments to the $CREPRC system service.

This procedure is illustrated in the example below.

NDESC: DESCRITOR FOR RESULT
NLLEN: LONG 63 LENGTH OF STRING RETURNED
NADDR: LONG NAME ADDRESS OF STRING
NAME: BLKB 63 DEVICE NAME STRING RETURNED
INPUT: DESCRIPTOR <SYS$INPUT> LOGICAL DEVICE NAME

$TRNLOG.S LOGNAME=INPUT,RSLLEN=NLLEN,RSLBUF=NDESC
BSBW ERROR BRANCH IF ERROR
CMPE NAME=\$"X1B FIRST BYTE AN ESCAPE?
BNEQ 10 NO, DON'T ADJUST
SUBL 4,NLEN SUBTRACT 4 FROM LENGTH
ADDL 4,NADDR ADD 4 TO ADDRESS
10*: $CREPRC.S ...,INPUT=NDESC,OUTPUT=NDESC...

When the subprocess executes, the logical names SYSSINPUT and SYSSOUTPUT are equated to the device name of the creating process's logical input device.

The subprocess can then use RMS to open the file for reading and/or writing. Or, the subprocess can use the Assign I/O Channel ($ASSIGN) system service to assign an I/O channel to this device for input/output operations by specifying the device name as the logical name SYSSOUTPUT. For example:

OUTPUT: DESCRIPTOR <SYS$OUTPUT> LOGICAL NAME DESCRIPTOR
OUTCHAN: BLKW 1 CHANNEL NUMBER OF OUTPUT DEVICE

$ASSIGN.S DEVNAME=OUTPUT,CHAN=OUTCHAN

Logical name translation is described in more detail in Section 3.3, "Logical Name Services." For more information on channel assignment for I/O operations, see Section 3.4, "Input/Output Services."

3.5.3.3 Disk and Directory Defaults for Created Processes - When you use the $CREPRC system service to create a process to execute an image, the system locates the image file within the context of the created process. The created process inherits the current default device and directory of its creator.

If you explicitly specify a device and/or directory in the file specification of the image file or the input, output or error equivalence names, then those files can be located within the context of the created process.

There is no way to define an alternative default device and/or directory at process creation. The created process can, however, define an equivalence for the logical device SYSDISK by calling the Create Logical Name ($CRELOG) system service. If the process is a subprocess, you can define an equivalence name in the group logical name table. The created process can also set its own default directory by calling the RMS Default Directory control routine. For details on how to call this routine, see the VAX-ll Record Management Services Reference Manual.
3.5.3.4 Controlling Resources of Created Processes - Ordinarily, when you create a subprocess, you need only assign it an image to execute and, optionally, SYS$INPUT, SYS$OUTPUT, and SYS$ERROR devices. The system provides default values for the process's privileges, resource quotas, execution modes, and priority. In some cases, you may want to specifically define these values. The arguments to the $CREPRC system service that control these characteristics are listed below, with considerations for their use. For details, see the argument descriptions of $CREPRC in Chapter 4.

1. PRVADR - this argument defines the privilege list for the created process. Normally, any process you create will have the same privileges that have been assigned to you by the system manager. In some circumstances, you may need to create a process that has a special privilege: but you must have the user privilege SETPRV to provide a subprocess with a privilege you do not have.

2. QUOTA - this argument defines the quota list for a subprocess. Since a subprocess receives a portion of its creator's quotas for timer queue entries, I/O buffers, and so on, you may want to control how much of each quota you want assigned to the subprocess. If you do not code this argument, the system defines default quotas for the subprocess.

3. STSFLG - the status flag is a set of bits that control some execution characteristics of the created process, including resource wait mode and process swap mode.

4. BASPRI - this argument sets the base execution priority for the created process. If not specified, it defaults to 2. If you want a subprocess to have a higher priority than its creator, you must have the user privilege ALTPRI to raise the priority level.

3.5.3.5 Detached Processes - The creation of a detached process is primarily a system function; the DETACH privilege controls the ability to create a detached process. The UIC argument to the $CREPRC system service defines whether a process is a subprocess or a detached process; it provides the created process with a user identification code (UIC). If you omit the UIC argument, the $CREPRC system service creates a subprocess that executes with your UIC.

3.5.4 Interprocess Control and Communication

Processes can be wholly independent, or they can be cooperative. You may develop an application that requires the concurrent execution of many programs. The following sections discuss the things you may consider when you develop such applications.

3.5.4.1 Restrictions on Process Creation and Control - There are three levels of process control privilege:

1. The creator of a subprocess can always issue control functions for that subprocess.
HOW TO USE SYSTEM SERVICES
PROCESS CONTROL SERVICES

2. The GROUP privilege is required to issue process control functions for other processes executing in the same group.

3. The WORLD privilege is required to issue process control functions for any process in the system.

Additional privileges are required to perform some specific functions, for example, to set a process's base priority to a higher level than that of the requestor.

3.5.4.2 Process Identification - In the examples shown in the preceding sections, the subprocesses are not identified: once created, the subprocesses execute according to the image name or the input stream specified, and are deleted when they complete execution. In many cases, however, you may want to be able to control the execution of a subprocess after it has been created. Or, detached processes that execute in the same group may want to communicate with one another or issue control functions. In these cases, the processes must be identified.

There are two levels of process identification:

1. Process identification number (PID). The system assigns this unique 32-bit number to a process when it is created. If you provide the PIDADR argument to the $CREPRC system service, the system returns the process identification at the location specified. You can then use the process identification number in subsequent process control services.

2. Process name. A process name is a 1- to 15-character text name string. You can assign a name to a process by coding the PRCNAM argument when you create it. You can then use this name to refer to the process in other system service calls.

For example, you might code a $CREPRC system service as follows:

ORION: DESCRIPTOR <ORION>  ;PROCESS NAME
ORIONID:
  .LONG 0  ;PROCESS ID RETURNED
  .
  .
  $CREPRC_S PRCNAM=ORION,PIDADR=ORIONID,...

The service returns the process identification in the longword at ORIONID. Now, you can use either the process name (ORION) or the process identification (ORIONID) to refer to this process in other system service calls.

A process can set or change its own name with the Set Process Name ($SETPRN) system service. For example, a process can set its name to CYGNUS as follows:

CYGNUS: DESCRIPTOR <CYGNUS>  ;NAME DESCRIPTOR
  .
  .
  $SETPRN_S PRCNAM=CYGNUS

3-42
HOW TO USE SYSTEM SERVICES
PROCESS CONTROL SERVICES

Most of the process control services accept either the PRCNAM or PIDADR arguments, or both. The process identification provides a more efficient means of identifying a process. Since it is only a longword in length, a system service can examine it more quickly.

When the PIDADR argument is coded and the specified address contains a 0, the services return the process identification. Thus, you can obtain the process identification for a process by issuing any control function, as long as you know the process name.

If neither argument is specified, the service is performed for the calling process. For a summary of the possible combinations of these arguments and an explanation of how the services interpret them, see Table 3-3.

Table 3-3
Process Identification

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>no</td>
<td>no</td>
<td>--</td>
<td>The process identification of the calling process is used. The process identification is not returned.</td>
</tr>
<tr>
<td>no</td>
<td>yes</td>
<td>zero</td>
<td>The process identification of the calling process is used and returned.</td>
</tr>
<tr>
<td>no</td>
<td>yes</td>
<td>process id</td>
<td>The process identification is used and returned.</td>
</tr>
<tr>
<td>yes</td>
<td>no</td>
<td>--</td>
<td>The process name is used. The process identification is not returned.</td>
</tr>
<tr>
<td>yes</td>
<td>yes</td>
<td>zero</td>
<td>The process name is used and the process identification is returned.</td>
</tr>
<tr>
<td>yes</td>
<td>yes</td>
<td>process id</td>
<td>The process identification is used and returned.</td>
</tr>
</tbody>
</table>

Process Naming within Groups: Process names are always qualified by their group number. The system maintains a table of all process names, and when a PRCNAM argument is specified in a process control service, the service searches for the process name specified and for a match on the group number, and fails if the specified process name does not have the same group number. This is true even if the calling process has world control privilege: to execute a process control service for a process that is not a subprocess and not in the caller's group, the requesting process must use a process identification.
Obtaining Information about Processes: The Get Job/Process Information ($GETJPI) system service allows a process to obtain information about itself or another process. For complete details about the $GETJPI system service, see the service description in Chapter 4.

Techniques for Interprocess Communication: There are several ways that processes can communicate:

- Common event flag clusters
- Logical name tables
- Mailboxes
- Global sections

Common Event Flag Clusters: Processes executing within the same group can use common event flag clusters to signal the occurrence or completion of particular activities. For details on event flags, event flag clusters, and an example of cooperating processes in the same group using a common event flag, see Section 3.1, "Event Flag Services."

Logical Name Tables: Processes executing in the same group can use the group logical name table to provide member processes with equivalence names for logical names. At least one member of the group must have the user privilege to place names in the group logical name table. For details on logical names and logical name tables, see Section 3.3, "Logical Name Services."

Mailboxes: Mailboxes can be used as virtual input/output devices to pass information, messages, or data among processes. For details on how to create and use mailboxes, with an example of cooperating processes using a mailbox, see Section 3.4, "Input/Output Services." Mailboxes may also be used to provide a creating process with a way to determine when and under what condition a created subprocess was deleted. See Section 3.5.7.2 for an example of a termination mailbox.

Global Sections: Global sections are disk files containing shareable code or data. Through the use of memory management services, these files can be mapped to the virtual address space of more than one process. In the case of a data file, cooperating processes can synchronize reading and writing the data in physical memory; as the data is updated, system paging results in the updated data being written directly back into the disk file. Global sections are described in more detail in Section 3.8.6, "Sections."
3.5.5 Process Hibernation and Suspension

There are two ways to temporarily halt the execution of a process: hibernation, performed by the Hibernate ($HIBER) system service, and suspension, performed by the Suspend Process ($SUSPEND) system service. The process can continue execution normally only after a corresponding Wake ($WAKE) system service, if it is hibernating; or after a Resume Process ($RESUME) system service, if it is suspended.

Process hibernation and suspension are compared in Table 3-4.

<table>
<thead>
<tr>
<th>Hibernation</th>
<th>Suspension</th>
</tr>
</thead>
<tbody>
<tr>
<td>Can only hibernate self</td>
<td>Can suspend self or another process, depending on privilege</td>
</tr>
<tr>
<td>Reversed by $WAKE system service</td>
<td>Reversed by $RESUME system service</td>
</tr>
<tr>
<td>Interruptible; can receive ASTs</td>
<td>Noninterruptible; cannot receive ASTs</td>
</tr>
<tr>
<td>Can wake self</td>
<td>Cannot resume self</td>
</tr>
<tr>
<td>Can schedule wakeup at an absolute time or at a fixed time interval</td>
<td>Cannot schedule resumption</td>
</tr>
<tr>
<td>Hibernate/wake complete quickly; require little system overhead</td>
<td>Requires system dynamic memory</td>
</tr>
</tbody>
</table>

3.5.5.1 Process Hibernation - The hibernate/wake mechanism provides an efficient way to prepare an image for execution and then place it in a wait state until it is needed. When the wake request is issued, the image is reactivated with little delay or system overhead.

For example, if you create a subprocess that must execute the same function repeatedly, but must execute immediately when it is needed, you could use the $HIBER and $WAKE system services as shown in Figure 3-13.

There is a variation of the $WAKE system service that schedules a wakeup for a hibernating process at a fixed time or at an elapsed (delta) time interval. This is the Schedule Wakeup ($SCHDWK) system service. Using the $SCHDWK service, a process can schedule a wakeup for itself before issuing a $HIBER call. For an example of how to use the $SCHDWK system service, see Section 3.6, "Timer and Time Conversion Services."

3-45
HOW TO USE SYSTEM SERVICES
PROCESS CONTROL SERVICES

Process GEMINI

ORION: DESCRIPTOR <ORION> ;SUBPROCESS NAME
FASTCOMP: DESCRIPTOR <COMPUTE.EXE> ;IMAGE

1 $CREPRC_S PRCNAM=ORION,IMAGE=FASTCOMP,... ;CREATE ORION
BSBW   ERROR ;CONTINUE

3 $WAKE_S PRCNAM=ORION ;WAKE ORION
BSBW   ERROR

$WAKE_S PRCNAM=ORION ;WAKE ORION AGAIN
BSBW   ERROR

Process ORION

FASTCOMP:

.WORD 0 2 ;ENTRY MASK
10$: $HIBER_S ;SLEEP
BSBW   ERROR ;PERFORM...

BRW 10$ ;BACK TO SLEEP

Notes:

1 Process GEMINI creates the process ORION, specifying the image name FASTCOMP.

2 The image FASTCOMP is initialized, and ORION issues the $HIBER system service.

3 At an appropriate time, GEMINI issues a $WAKE request for ORION. ORION continues execution following the $HIBER service call. When it finishes its job, ORION loops back to repeat the $HIBER call and to wait for another wakeup.

Figure 3-13 Process Hibernation

Hibernating processes can be interrupted by Asynchronous System Traps (ASTs), as long as AST delivery is enabled. The process can issue a $WAKE for itself in the AST service routine, and continue execution following the execution of the AST service routine. For a description of ASTs, and how to use them, see Section 3.2, "AST (Asynchronous System Trap) Services."
3.5.5.2 Alternate Methods of Hibernation - Two additional techniques you can use to cause a process to hibernate are:

- Code the STSFLG argument for the $CREPRC system service, setting the bit that requests $CREPRC to place the created process in a state of hibernation as soon as it is initialized.

- Specify the /DELAY, /SCHEDULE, or /INTERVAL qualifiers of the RUN command when you execute the image from the command stream.

When you use the first method, the creating program image can control, the system services described here and in Section 3.6, when to wake the created process.

When you use the RUN command, the qualifiers listed above control when the process will be awakened.

If the image to be executed does not, itself, call the $HIBER system service, the image is placed in a state of hibernation whenever it issues a RET instruction. Each time it is reawakened, it begins executing at its entry point. If the image does not call $HIBER, then it begins executing at either the point following the call to $HIBER or at its entry point (if it issues a RET instruction) each time it is awakened.

If wakeup requests are scheduled at time intervals, the image can be terminated with the Delete Process ($DELPRC) or Force Exit ($FORCEEX) system services, or from the command level, with the STOP command. The $DELPRC and $FORCEEX system services are described later in this section. The RUN and STOP commands are described in the VAX/VMS Command Language User's Guide.

These techniques allow you to code programs that can be executed a single time, on request, or cyclically, depending on a particular set of circumstances. Note that the program must ensure the integrity of data areas that are modified during its execution, as well as the status of opened files.

3.5.5.3 Suspension - Using the Suspend Process ($SUSPND) system service, a process can place itself or another process into a wait state similar to hibernation. Suspension, however, is a more pronounced state of hibernation. A suspended process cannot be interrupted by ASTs, and can resume execution only after another process issues a Resume Process ($RESUME) system service for it. If ASTs were queued for the process while it was suspended, they are delivered when the process resumes execution.

3.5.6 Image Exit

When the image executing in a process completes normally, the operating system performs a variety of image rundown functions. If the image was executed by the command interpreter, image rundown prepares the process for the execution of another image. If the image was not executed by the command interpreter -- for example, if it was executed by a subprocess -- the rundown readsies the process for deletion.
These exit activities are also initiated when an image completes abnormally, as a result of any of the following:

1. Specific error conditions caused by improper specifications when a process was created. For example, if an invalid device name is specified for SYS$INPUT, SYS$OUTPUT, or SYS$ERROR logical names, or if an invalid or nonexistent image name is specified, the error condition is noted within the context of the created process.

2. An exception condition during execution of the image. When an exception condition occurs, any user-specified condition handlers receive control to handle the exception. If not, a system-declared condition handler receives control, and it initiates exit activities for the image. Condition handling is described in Section 3.7, "Condition Handling Services."

3. A Force Exit ($FORCEx) system service issued on behalf of the process by another process.

3.5.6.1 Image Rundown Activities - The operating system performs image rundown functions that release system resources that a process obtained while executing in user mode. These activities are listed below.

- Exit handlers declared from user mode, if any, are called, and the exit control blocks are released. (Exit handlers are described in Section 3.5.6.3.)
- Common event flag clusters are disassociated.
- User mode ASTs that are queued but have not been delivered are deleted, and ASTs are enabled for user mode.
- I/O channels are deassigned and any outstanding I/O requests on the channels are canceled.
- All devices allocated to the process at user mode are deallocated.
- Timer-scheduled requests, including wakeup requests, are canceled.
- Logical names in the process logical name table entered in user mode are deleted (logical names entered from the command stream in supervisor mode are not deleted).
- Exception vectors declared in user mode, compatibility mode handlers, and change mode to user handlers are reset.
- System service failure exception mode is disabled.
- Memory pages occupied by the image are deleted and the process's working set size limit is readjusted to its default value.

3.5.6.2 The $Exit System Service - To initiate the rundown activities described above, the system calls the Exit ($EXIT) system service on behalf of the process. In some cases, a process can call $EXIT to terminate the image itself, for example, if an unrecoverable error occurs. This is not, however, recommended programming practice.
The $EXIT system service accepts a status code as an argument. If you use $EXIT to terminate image execution, you can use this status code argument to pass information about the completion of the image. If an image does not call $EXIT, the current value in R0 is passed as the status code when the system calls $EXIT.

This status code is used as follows:

- The command interpreter uses the status code to display an error message when it receives control following image rundown.
- If the image has declared an exit handler, the status code is written in the address specified in the exit control block.
- If the process was created by another process, and the creator has specified a mailbox to receive a termination message, the status code is written in the termination message when the process is deleted.

The use of exit handlers and termination messages requires additional coding considerations. These considerations are discussed in greater detail below.

3.5.6.3 Exit Handlers - Exit handlers are routines that can perform image-specific cleanup or rundown operations. For example, if an image uses system memory to buffer data, an exit handler can ensure that the data is not lost when the image exits as the result of an error condition.

To establish an exit handling routine, you must set up an exit control block, and specify the address of the control block on the Declare Exit Handler ($DCLEXH) system service. Exit handlers are called using standard calling conventions; you can provide arguments to the exit handler in the exit control block. The first argument in the control block argument list must specify the address of a longword for the system to write the status code from $EXIT.

If an image declares more than one exit handler, the control blocks are linked together on a last-in, first-out basis. After an exit handler has been called and returns control, the control block is removed from the list. Exit control blocks can also be removed prior to image exit with the Cancel Exit Handler ($CANEXH) system service.

Exit handlers can also be declared from system routines executing in supervisor or executive modes. These exit handlers are also linked together, and receive control after exit handlers declared from user mode have been executed.
HOW TO USE SYSTEM SERVICES  
PROCESS CONTROL SERVICES  

Figure 3-14 shows an example of an exit handling routine.

EXITBLOCK:  
.LONG 0  
.LONG EXITRTN  
.LONG 1  
.LONG STATUS  
STATUS: .BLKL 1  

;EXIT CONTROL BLOCK  
;SYSTEM USES THIS FOR POINTER  
;ADDRESS OF EXIT HANDLER  
;NUMBER OF ARGS FOR HANDLER  
;ADDRESS TO RECEIVE STATUS CODE  
;STATUS CODE FROM $EXIT  

;PEGASUS: .WORD ^M<R2>R3>  
$DCLEXS DESBLK=EXITBLOCK  
$ENTRY MASK FOR PEGASUS  
$DECLARE EXIT HANDLER  
BSEW ERROR  

;END OF MAIN ROUTINE  
EXITRTN:  
.WORD ^M<R2>  
CMPL STATUS,$SS$NORMAL  
;ENTRY MASK  
;NORMAL EXIT?  
;YES, FINISH  
;NO, CLEAN UP  

10$: RET  
;FINISHED  

Notes:

1 EXITBLOCK is the exit control block for the exit handler EXITRTN. The third longword indicates the number of arguments to be passed; in this example, only one argument is passed. This is the address of a longword for the system to store the return status code; this argument must be provided in an exit control block.

2 The $DCLEXS system service call designates the address of the exit control block, thus declaring EXITRTN as an exit handler.

3 EXITRTN checks the status code. If this is a normal exit, EXITRTN returns control. Otherwise, it handles the error condition.

Figure 3-14 Example of an Exit Handler

3.5.6.4 Forced Exit - The Force Exit ($FORCEX) system service provides a way for a process to initiate image rundown for another process. For example, the following call to $FORCEX causes the image executing in the process CYGNUS to exit:

CYGNUS: DESCRIPTOR <CYGNUS>  
;PROCESS NAME  

$FORCEX_S PRCNAM=CYGNUS  

The $FORCEX system service uses the AST mechanism to cause the image to exit. If the process CYGNUS has disabled AST delivery, the image cannot be forced to exit until CYGNUS reenables the delivery of ASTs. AST delivery, and how it is disabled and reenabled, is described in Section 3.2.
3.5.7 Process Deletion

Process deletion completely removes a process from the system. Deletion occurs as a result of any of the following conditions:

- The command stream contains a LOGOUT command or an end-of-file.
- An image specified by $CREPRC exits.
- A process issues a STOP command or executes an image that calls the Delete Process ($DELPRC) system service.

When the system is called to delete a process as a result of any of the above conditions, it first locates all subprocesses, searching hierarchically. Then, beginning with the lowest process in the hierarchy, and completing with the topmost process, each of the following are performed:

- The image executing in the process is run down. System resources are released, and, if this is a subprocess, quotas are returned to the creator of the process. The image run-down that occurs during process deletion is the same as that described in Section 3.5.6.1. When a process is deleted, however, the run-down releases all system resources, including those acquired from access modes other than user mode.
- Resource quotas are released to the creating process, if it is a subprocess.
- If the creating process specified a termination mailbox, a message indicating that the process is being deleted is sent to the mailbox. For detached processes created by the system, the termination message is sent to the system job controller.
- The control region of the process's virtual address space is deleted. (The control region consists of memory allocated and used by the system on behalf of the process.)
- All system-maintained information about the process is deleted.

Figure 3-15 illustrates the flow of events from image exit through process deletion.
Figure 3-15 Image Exit and Process Deletion
3.5.7.1 The Delete Process System Service - A process can delete itself or another process at any time, depending on the restrictions outlined in Section 3.5.4.1. The Delete Process ($DELPRC) system service deletes a process. For example, if a process has created a subprocess named CYGNUS, it can delete CYGNUS as shown below:

`CYGNUS: DESCRIPTOR <CYGNUS>

:  

$DELPRC_S PRCNAM=CYGNUS`

Since a subprocess is automatically deleted when the image it is executing terminates (or when the command stream for the command interpreter reaches end-of-file), you do not normally need to issue the $DELPRC system service explicitly.

As an alternative to deleting a process, you can use the Force Exit ($FORCEX) system service to force the exit of the image executing in a process. If the $FORCEX system service is used, any exit handlers that are declared for the image are executed during the image rundown. Thus, if the process is using the command interpreter, it is not deleted, but can run another image. Moreover, since the $FORCEX system service uses the AST mechanism, the exit cannot be performed if the process being forced to exit has disabled the delivery of ASTs.

3.5.7.2 Termination Mailboxes - A termination mailbox provides a process with a way of determining when, and under what conditions, a process that it has created is being deleted. The Create Process ($CREPRC) system service accepts the unit number of a mailbox as an argument. When the created process is deleted, the mailbox receives a termination message.

The first word of the termination message contains the symbolic constant, MSG$_DELPROC, which indicates that it is a termination message. The remainder of the message contains system accounting information used by the job controller, and is in fact identical to the first part of the accounting record sent to the system accounting log file. The complete format of the termination message is provided with the description of the $CREPRC system service in Chapter 4.

The creating process can, if necessary, determine the process identification of the process being deleted from the I/O status block posted when the message is received in the mailbox. The second longword of the IOSB contains the process identification of the process that is being deleted.

Figure 3-16 illustrates a complete sequence of process creation, with a termination mailbox. The Create Mailbox and Assign Channel ($CREMBX) and Queue I/O Request ($QIO) system services are described in greater detail in Section 3.4.
HOW TO USE SYSTEM SERVICES
PROCESS CONTROL SERVICES

```
EXCHAN: .BLKW 1
EXITBUF: .LONG ENDBUF-BBUF
         .LONG BBUF
BBUF:   .BLKB DIB*K_LENGTH
ENDBUF:
EXITMSG: .BLKB ACC*K_TERMLEN
MBXIOSB: .BLKW 1
MBLEN:   .BLKW 1
MBPID:   .BLKL 1
LYRAPID:
LYREXE:  DESCRIPTEOR <LYRA,EXE>

1 $CREMBX_S _CHAN=EXCHAN,MAXMSG=$120,PROMSK=$0,BUFQUO=$240
    ;CREATE MAILBOX
BSBW  ERROR
2 $GETCNCH_S _CHAN=EXCHAN,PRIBUF=EXITBUF
    ;GET MAILBOX INFO
BSBW  ERROR
3 $CREPRC_S IMAGE=LYREXE,PIDADR=LYRAPID, -
    ;CREATE SUBPROCESS
    MBXUNIT=BBUF+DIB*W_UNIT ;SPECIFY TERMINATION MAILBOX
BSBW  ERROR
4 $QIO_S    CHAN=EXCHAN,FUNC=#ID$_READVBLK, -
    ;QIO TO MAILBOX
    ASTADR=EXITAST,IOSB=MBXIOSB,P1=EXITMSG,P2=ACC*K_TERMLEN
BSBW  ERROR
    ;CONTINUE EXECUTION
RET
EXITAST:  ;AST ROUTINE FOR TERMINATION MSG
6 .WORD 0
    ;ENTRY MASK
CMPW MBXIOSB,#$*_NORMAL
BNEQ 20$ ;I/O SUCCESSFUL?
CMPW EXITMSG+ACC*W_MSGTY+*MSG*DELPROC ;IS IT A TERMINATION MSG?
BNEQ 20$ ;NO, SOME THING ELSE
CMPL LYRAPID,MBPID
BNEQ 20$ ;IS IT LYRA?
CMPL EXITMSG+ACC*L_FINALSTS,#$*_NORMAL ;DELETED NORMALLY?
BEGL 10$ ;YES, RETURN
          ;NO, RESPOND TO ERROR IN LYRA
10$: RET ;AST ROUTINE FINISHED
20$: ;HANDLE ALL OTHER CONDITIONS
```

Figure 3-16 Using a Termination Mailbox
HOW TO USE SYSTEM SERVICES
PROCESS CONTROL SERVICES

Notes on Figure 3-16:

1. The Create Mailbox and Assign Channel ($CREMBX) system service creates the mailbox, and returns the channel number at EXCHAN.

2. The Get I/O Channel Information ($GETCHN) system service returns information about the mailbox. The information returned in the buffer can be referred to by the symbolic offsets defined in the $DIBDEF macro.

3. The Create Process ($CREPRC) system service creates a process to execute the image LYRA.EXE, and returns the process identification at LYRAPID. The MBXUNIT argument refers to the unit number of the mailbox, obtained from the buffer BBUF by using the symbolic offset DIB$W_UNIT.

4. The Queue I/O Request queues a read request to the mailbox, specifying an AST service routine to receive control when the mailbox receives a message and the address of a buffer to receive the message. The information in the message can be accessed by the symbolic offsets defined in the $ACCDEF macro. The process continues executing.

5. When a message is received in the mailbox, the AST service routine, EXITAST, receives control. Since this mailbox can be used for other interprocess communication, the AST routine checks: 1) for successful completion of the I/O operation by examining the first word in the IOSB, 2) that the message received is a termination message by examining the message type field in the termination message at the offset ACC$W_MSGTYPE, 3) the process identification of the process that has been deleted by examining the second longword of the IOSB, and 4) the completion status of the process by examining the status field in the termination message at the offset ACC$L_FINALSTS.

In this example, the AST service routine performs special action when the subprocess is deleted. All other messages or error conditions cause a branch to the label 20$.
3.6 TIMER AND TIME CONVERSION SERVICES

Many applications require the scheduling of program activities based on clock time. In VAX/VMS, an image can schedule events for a specific time of day, or after a specified time interval. Timer services:

- Schedule the setting of an event flag or the queuing of an asynchronous system trap (AST) for the current process, and cancel a pending request that has not yet been honored.
- Schedule a wakeup request for a hibernating process, and cancel a pending wakeup request that has not yet been honored.

The timer services require you to specify the time in a unique 64-bit format. Time conversion services:

- Obtain the current date and time in an ASCII string or in system format
- Convert an ASCII string into the system time format
- Convert a system time value into an ASCII string
- Convert the time from system format to integer values

This section describes the system time format and the services that use it, with examples of scheduling program activities using the timer services.

3.6.1 The System Time Format

VAX/VMS maintains the current date and time (using a 24-hour clock) in 64-bit format. The time value is a binary number in 100-nanosecond units offset from the system base date and time, which is 00:00 o'clock, November 17, 1858. All time values passed to system services must also be in 64-bit format. A time value can be expressed as:

- An absolute time which is a specific date and time of day. Absolute times are always positive values.
- A delta time which is a future offset (number of hours, minutes, seconds, and so on) from the current time. Delta times are always expressed as negative values.

You can also specify the address of a time value as 0; in this case the system will always supply the current date and time by default.

3.6.2 The Current Date and Time

The Convert Binary Time to ASCII String ($ASCTIM) system service converts a time in system format to an ASCII string and returns the string in a 24-byte buffer. If you want to obtain the current time in ASCII, code the $ASCTIM system service as follows:

---

1 This is the Smithsonian base date and time for the astronomical calendar.
HOW TO USE SYSTEM SERVICES
TIMER AND TIME CONVERSION SERVICES

ATIMENOW:
.LONG 20$-10$
.LONG 10$
10$:
20$:

$\#$ASCTIM_S TIMBUF=ATIMENOW, - $\#$GET CURRENT TIME
TIMLEN=ATIMENOW

The string returned by the service in the buffer ATIMENOW has the format:

dd-mm-yyyy hh:mm:ss.cc

where dd is the day of the month, mmm is the month (a 3-character alphabetic abbreviation), yyyy is the year, and hh:mm:ss.cc is the time in hours, minutes, seconds, and hundredths of seconds. The TIMLEN argument requests the system to place the length of the string returned in the first word of the descriptor.

The current time can also be obtained in system format with the Get Time ($SGETTIM$) system service, which places the time in a quadword buffer. For example:

TIME: .BLKB 1

$\#$BUFFER FOR TIME

$\#$GETTIME

This call to $SGETTIM$ returns the current date and time system format in the quadword buffer TIME.

3.6.3 Obtaining an Absolute Time in System Format

The converse of the $ASCTIM$ system service is the Convert ASCII String to Binary Time ($SBINTIM$) system service. You provide the service with the time in the ASCII format shown above, and the service converts the string to a time value in 64-bit format. You can then use this returned value as input to a timer scheduling service.

When you code the ASCII string buffer, you can omit any of the fields, and the service uses the current date or time value for the field. Thus, if you want a timer request to be date independent, you could format the input buffer for the $SBINTIM$ service as shown below. The two hyphens that are normally embedded in the date field must be included; at least one blank must precede the time field.

ANoon: DESCRIPTOR <-- 12:00:00.00>  $\#$ASCII 12 NOON
BNOON: .BLKB 1 $\#$BUFFER FOR BINARY 12 $\#$NOON

$\#$SBINTIM_S TIMBUF=ANoon,TIMADR=BNOON $\#$CONVERT TIME

When the $SBINTIM$ service completes, a 64-bit time value representing "noon today" is returned in the quadword at BNOON.
3.6.4 Obtaining a Delta Time in System Format

The $BINTIM system service also converts ASCII strings to delta time values to be used as input to timer services. The buffer for delta time ASCII strings has the format:

```
dddh:mm:ss.cc
```

The first field, indicating the number of days, must be specified as 0 if you are coding a "today" delta time.

The following example shows how to use the $BINTIM service to obtain a delta time in system format.

```
ATENMIN: DESCRIPTOR <0 00:10:00.00> ;ASCII TEN MINUTES
BTENMIN:
  .BLKQ 1 ;BUFFER FOR BINARY TEN MINUTES

$BINTIM_S TIMBUF=ATENMIN,TIMADR=BTENMIN ;CONVERT TIME
```

You can also specify approximate delta time values at assembly time, using two MACRO .LONG directives to represent a time value in terms of 100-nanosecond units. The arithmetic is based on the formula:

```
1 second = 10 million * 100 nanoseconds
```

For example, the following statement defines a delta time value of 5 seconds:

```
FIVESEC: .LONG -10*1000*1000*5,-1 ;FIVE SECONDS
```

The value 10 million is expressed as `10*1000*1000` for readability. Note that the delta time value is negative.

If you use this notation, however, you are limited to the maximum number of 100-nanosecond units that can be expressed in a longword. In terms of time values, this is somewhat more than 7 minutes.

3.6.5 Timer Requests

Timer requests made with the Set Timer ($SETIMR) system service are queued, that is, they are ordered for processing according to their expiration times. The TQELM quota controls the number of entries a process can have pending in this timer queue.

When you code the $SETIMR system service, you can specify either an absolute time or a delta time value. Depending on how you want the request processed, you can specify either or both of the following:

- The number of an event flag to be set when the time expires. If you do not specify an event flag, the system sets event flag 0.
- The address of an AST service routine to be executed when the time expires.

Optionally, you can specify a request identification for the timer request. You can use this identification to cancel the request, if necessary. The request identification is passed to the AST service
HOW TO USE SYSTEM SERVICES
TIMER AND TIME CONVERSION SERVICES

routine, if one is specified, as the AST parameter so the AST service routine can identify the timer request.

Figure 3-17 shows examples of timer requests using event flags and ASTs. Event flags and event flag services are described in more detail in Section 3.1, "Event Flag Services." ASTs are described in more detail in Section 3.2, "AST (Asynchronous System Trap) Services."

Example 1: Setting an Event Flag

```
WAITIME:
.LONG -10*1000*1000*30,-1       ;30 SECOND WAIT TIME

1 $SETIMR_S EFN=4, DAYTIM=WAITIME ;SET TIMER
   BSBW ERROR
2 $WAITFR_S EFN=4                  ;WAIT 30 SECONDS
   BSBW ERROR
```

Notes on Example 1:

1. The call to $SETIMR requests that event flag 4 be set in 30 seconds (expressed in the quadword WAITIME).

2. The Wait for Single Event Flag ($WAITFR) system service places the process in a wait state until the event flag is set. When the timer expires, the flag is set and the process continues execution.

Figure 3-17 Timer Requests
Example 2: Using an AST Service Routine

ANON:  DESCRIPTOR <- 12:00:00.00>  #ASCII NOON
BNOON: .BLKD 1  #BINARY NOON

1 $BINTIM_S TIMBUF=ANON,TIMADR=BNOON  #CONVERT TIME
BSBW ERROR
2 $SETIMR_S DAYTIM=BNOON,ASTADR=ASTSERV,REQIDT=$12
BSBW ERROR

ASTSERV:

3 .WORD 0  #ENTRY MASK
CMPL $12,4(AP)  #CHECK AST PARAMETER
BEQ 10$  #GO TO NOON ROUTINE

10$:

;SERVICE NOON REQUEST

RET

Notes on Example 2:

1 The call to $BINTIM converts the ASCII string representing
12:00 noon to system format. The value returned in BNOON is
used as input to the $SETIMR system service.

2 The AST routine specified in the $SETIMR request will be
called when the timer expires, that is, at 12:00 noon. The
REQIDT argument identifies the timer request. The process
continues execution; when the timer expires, it is
interrupted by the delivery of the AST. Note that if the
current time of day is past noon, the timer expires
immediately.

3 This AST service routine checks the parameter passed by the
REQIDT argument and determines, in this example, that it must
service the 12:00 noon timer request. When the AST service
routine completes, the process continues execution at the
point of interruption.

Figure 3-17 (Cont.) Timer Requests

3.6.5.1 Canceling Timer Requests - The Cancel Timer Request ($SCANTIM)
system service cancels timer requests that have not yet been
processed. The entries are removed from the timer queue.
Cancellation is based on the request identification given in the timer
request. For example, to cancel the request illustrated in Example 2
of Figure 3-17, you would code:

$CANTIM_S REQIDT=$12

If you assign the same identification to more than one timer request,
all requests with that identification are canceled. If you do not
specify the REQIDT argument, all your requests are canceled.
3.6.6 Scheduled Wakeups

Figure 3-17 showed a process placing itself in a wait state for a period of time using the $SETIMR and $WAITFR services. Another way for a process to make itself inactive is by hibernating. A process hibernates by issuing the Hibernate (SHIBER) system service; hibernation is reversed by a wakeup request, which can be effected immediately with the $WAKE system service, or scheduled with the Schedule Wakeup ($SCHDWK) system service.

The following example shows a process scheduling a wakeup for itself prior to hibernating:

```
ATENSEC: DESCRIPTOR <0 00:00:10.00>  $10 SECOND WAIT TIME
BTENSEC:
  .BLKQ 1  $BINARY TEN SECONDS
  
  $BINTIM_S TIMBUF=ATENSEC,TIMADR=BTENSEC  $CONVERT TIME
  $SCHDWK_S DAYTIM=BTENSEC  $SCHEDULE WAKE
  $HIBER_S  $SLEEP TEN SECONDS
```

Hibernation and wakeup are described in more detail in Section 3.5, "Process Control Services." Note that a suitably privileged process can wake or schedule a wakeup for another process; thus, cooperating processes can synchronize activity using hibernation and scheduled wakeups. Moreover, when you code a $SCHDWK system service, you can specify that the wakeup request be repeated at fixed time intervals.

3.6.6.1 Canceling Scheduled Wakeups - Scheduled wakeup requests that are pending but have not yet been processed can be canceled with the Cancel Wakeup ($CANWAK) system service.

The following example shows the scheduling of wakeup requests for a process, CYGNUS, and the subsequent cancellation of the wakeups. The $SCHDWK system service in this example specifies a delta time of one minute and an interval time of one minute; the wakeup is repeated every minute until the requests are canceled.

```
CYGNUS: DESCRIPTOR <CYGNUS>  $PROCESS NAME
INTERVAL:
  .LONG -10*1000*1000*60,-1  $ONE MINUTE
  
  $SCHDWK_S PRCNAM=CYGNUS,Daytim=Interval,Reptim=Interval
  
  $CANWAK_S PRCNAM=CYGNUS  $CANCEL WAKES
```

3.6.7 Numeric and ASCII Time

The Convert Binary Time to Numeric Time ($NUMTIM) system service converts a time in the system format into binary integer values. The service returns each of the components of the time (year, month, day, hour, and so on) into a separate word of a seven-word buffer. The $NUMTIM system service and the format of the information returned are described in Chapter 4.
When you need the time formatted into ASCII for inclusion in an output string, you can use the $ASCTIM system service. The $ASCTIM service accepts as an argument the address of a quadword that contains the time in system format and returns the date and time in ASCII format.

If you want to include the date and time in a character string that contains additional data, you can format the output string with the Formatted ASCII Output ($FAO) system service. The $FAO system service converts binary values to ASCII representations, and substitutes the results in character strings according to directives supplied in an input control string. Among these directives are $%T and $%D, which convert a quadword time value to an ASCII string and substitute the result in an output string. For examples of how to do this, see the discussion of $FAO in Chapter 4.
3.7 CONDITION HANDLING SERVICES

Exceptions are hardware- or software-detected conditions that interrupt the execution of an image. Exceptions are caused by such things as arithmetic overflow or underflow conditions, or reserved opcode or operand faults.

Condition handlers are procedures that are given control when an exception condition occurs. If you determine that a program needs to be informed of particular exception conditions so that it can perform corrective action, you may want to code a condition handling routine. This routine, or condition handler, then receives control when any type of exception occurs.

If an exception occurs, and no condition handler exists, the default condition handler established by the command interpreter is given control. This handler issues a descriptive message and performs an exit on behalf of the image that incurred the exception.

This section describes how the condition handling mechanism in VAX/VMS works, and explains how to write a condition handler.

3.7.1 Types of Exception

Exception conditions can be generated by:

- Hardware
- Software
- System service failures

Hardware-generated exceptions always result in conditions that require special action if program execution is to continue. A list of hardware exceptions is given in Table 3-5.

Some software routines can generate exception conditions; these may be warning or error conditions. (These software conditions are documented with the descriptions of any software that cause them.)

Software exceptions can also occur when an error or severe error status is returned from a call to a system service. You can choose to handle error returns from system services by using the condition handling mechanism rather than other error checking methods. If you want exceptions generated by service failures, you must enable system service failure exception mode with the Set System Service Failure Mode ($SETSFM) system service. For example:

$SETSFM_S ENBFLG=4

System service failure exception mode is initially disabled, and may be enabled or disabled at any time during the execution of an image.
HOW TO USE SYSTEM SERVICES
CONDITION HANDLING SERVICES

3.7.1.1 Change Mode and Compatibility Mode Handlers - There are two
types of hardware exception that can be handled in a special way,
bypassing the normal condition handling mechanism described in this
chapter. These are:

- Traps caused by change mode to user or change mode to
  supervisor instructions
- Compatibility mode faults

You can use the Declare Change Mode or Compatibility Mode Handler
($DCLCMH) system service to establish procedures to receive control
when one of these conditions occurs. The $DCLCMH system service is
described in Chapter 4.

3.7.2 How to Specify Condition Handlers

You can establish condition handlers to receive control in the event
of an exception in two ways:

1. By specifying the address of the entry mask of a condition
   handler in the first longword of a procedure call frame

2. By establishing exception vectors with the Set Exception
   Vector ($SETEXV) system service

The first of these methods is the most common way to specify a
condition handler for a particular image. It is also the most
efficient way in terms of declaration. You only have to use a single
move address instruction to place the address of the condition handler
in the longword pointed to by the current frame pointer (FP). For
example:

    MOVAL    HANDLER,(FP)

Each procedure on the call stack can declare a condition handler.

The $SETEXV system service allows you to specify addresses for a
primary exception vector, a secondary exception vector, and a last
chance exception vector. Vectors may be specified for each access
mode. The primary exception vector is reserved for the debugger.

An address of 0, in the first longword of a procedure call frame, or
in an exception vector, indicates that no condition handler exists for
the respective vector or call frame.

3.7.3 The Exception Dispatcher

When an exception condition occurs, control is passed to the operating
system's exception dispatching routine. The exception dispatcher

3-64
searches for a condition handling routine using the following search order:

1. The primary exception vector for the access mode at which the program was executing when the exception occurred.

2. The secondary exception vector for the access mode at which the program was executing when the exception occurred.

3. The condition handler address specified in the procedure call stack of the access mode at which the program was executing when the exception occurred. Call frames on the stack are scanned backwards, using the saved frame pointer in each call frame to refer to the previous call frame.

4. The last chance exception vector for the access mode at which the program was executing when the exception occurred.

The search is terminated when the dispatcher finds a condition handler. If the dispatcher cannot find a user-specified condition handler, it calls the default condition handler established by the command interpreter, if the image was initiated by the command interpreter. The default handler issues a message and either continues program execution or performs an exit on behalf of the process, depending on whether the condition was a warning or an error condition, respectively.

The search can also be terminated when the dispatcher detects a saved frame pointer containing a 0 (that is, it reaches the end of the stack), or when an access violation occurs. In these cases, the system performs an exit for the process, with the return status code SS$_NOHANDLER indicating "absence of condition handler" (for a 0 frame pointer) or SS$_ACCVIO indicating "bad stack" (for an access violation).

Figure 3-18 illustrates the exception dispatcher's search of the call stack for an exception handler.
Figure 3-18 Search of Stack for Condition Handler

Notes on Figure 3-18:

1. The illustration of the call stack indicates the calling sequence: Procedure A called Procedure B, and Procedure B called Procedure C. Procedure A established a condition handler.

2. An exception condition occurs while Procedure C is executing. The exception dispatcher searches for a condition handler.

3. After checking for a condition handler declared in the exception vectors (assume that none has been specified for this process), the dispatcher looks at the first longword of Procedure C's call frame. A value of 0 indicates that no condition handler has been specified. The dispatcher locates the call frame for Procedure B by using the frame pointer (FP) in Procedure C's call frame. Again, it finds no condition handler, and locates Procedure A's call frame.

4. The dispatcher locates and gives control to HANDLER A.
3.7.4 The Argument List Passed to a Condition Handler

When the dispatcher finds a condition handler, it passes control to it using a CALLG instruction. The argument list passed to the condition handler is constructed on the stack and consists of the addresses of two argument arrays, as illustrated in Figure 3-19; these arguments are described in detail in Sections 3.7.4.1 and 3.7.4.2.

![Diagram of Argument List and Arrays](image)

You can define symbolic names to refer to these arguments using the $CHFDEF macro instruction. The symbolic names are:

<table>
<thead>
<tr>
<th>Symbolic Offset</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHFSL_SIGARGLST</td>
<td>Address of signal array</td>
</tr>
<tr>
<td>CHFSL_MCHARGLST</td>
<td>Address of mechanism array</td>
</tr>
<tr>
<td>CHFSL_SIG_ARGS</td>
<td>Number of signal arguments</td>
</tr>
<tr>
<td>CHFSL_SIG_NAME</td>
<td>Condition name</td>
</tr>
<tr>
<td>CHFSL_SIG_ARG1</td>
<td>First signal-specific argument</td>
</tr>
<tr>
<td>CHFSL_MCH_ARGS</td>
<td>Number of mechanism arguments</td>
</tr>
<tr>
<td>CHFSL_MCH_FRAME</td>
<td>Estabisher frame address</td>
</tr>
<tr>
<td>CHFSL_MCH_DEPTH</td>
<td>Frame depth of estabisher</td>
</tr>
<tr>
<td>CHFSL_MCH_SAVR0</td>
<td>Saved register 0</td>
</tr>
<tr>
<td>CHFSL_MCH_SAVR1</td>
<td>Saved register 1</td>
</tr>
</tbody>
</table>

Figure 3-19 Argument List and Arrays Passed to Condition Handler
3.7.4.1 **Signal Array Arguments** - The signal array contains values describing the exception condition. These are:

1. **Condition name** -- the symbolic value assigned to the specific exception condition. The possible conditions, and their symbolic definitions, are listed in Table 3-5.

2. **Additional arguments** -- specific information relating to the condition. Table 3-5 also shows the additional arguments provided with each exception condition.

3. **PC** -- the program counter at the time of the exception. Depending on the type of exception (fault or trap), this can be the address of the instruction that caused the exception, or the following instruction, respectively.

4. **PSL** -- the processor status longword at the time of the exception.

3.7.4.2 **Mechanism Array Arguments** - The mechanism array describes the context in which the condition occurred. The arguments supplied are:

1. **Establisher frame** -- the frame pointer (FP) register image of the call frame that established the condition handler. This is the address of the longword containing the condition handler address. For example, if the call stack is as shown in Figure 3-18, this argument points to the call frame for Procedure A. This value can be used to display local variables in the procedure that established the condition handler, if the variables are at known offsets from the FP of the procedure.

2. **Depth** -- the frame number of the procedure that established the condition handler, relative to the frame of the procedure that incurred the exception. The depth is determined as follows:

<table>
<thead>
<tr>
<th>Depth</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-3</td>
<td>Condition handler was established in the last chance exception vector</td>
</tr>
<tr>
<td>-2</td>
<td>Condition handler was established in the primary exception vector</td>
</tr>
<tr>
<td>-1</td>
<td>Condition handler was established in the secondary exception vector</td>
</tr>
<tr>
<td>0</td>
<td>Condition handler was established by the frame that was active when the exception occurred</td>
</tr>
<tr>
<td>1</td>
<td>Condition handler was established by the caller of the frame that was active when the exception occurred</td>
</tr>
<tr>
<td>2</td>
<td>Condition handler was established by the caller of the caller of the frame that was active when the exception occurred</td>
</tr>
<tr>
<td>..</td>
<td>and so on.</td>
</tr>
</tbody>
</table>

3-68
HOW TO USE SYSTEM SERVICES
CONDITION HANDLING SERVICES

For example, if the call stack is as shown in Figure 3-18, the depth argument passed to HANDLERA would have a value of 2.

The condition handler can use this argument to determine whether it wants to handle the condition. For example, the handler may not want to handle the condition if the condition did not occur in the establisher frame.

3. R0 -- the contents of register 0 when the exception condition occurred.

4. R1 -- the contents of register 1 when the exception condition occurred.

3.7.5 Courses of Action for the Condition Handler

After the condition handling routine determines the nature of the exception, it can take one of the following courses of action:

1. Continue

The condition handler may or may not be able to fix the problem but the program can continue execution. The handler places the return status value SS$CONTINUE in R0 and issues a RET instruction to return control to the dispatcher. The exception dispatcher returns control to the procedure that incurred the exception, at the instruction that caused the exception. If the exception was a fault, the instruction that caused it is reexecuted; if the exception was a trap, control is returned at the instruction following the one that caused it. (In the case of a trap, the instruction causing the trap can sometimes be re-executed by subtracting the length of the instruction from the PC in the signal array.)

2. Resignal

The handler cannot fix the problem, or this condition is one that it does not handle. It places the return status value SS$RESIGNAL in R0 and issues a RET instruction to return control to the exception dispatcher. The dispatcher resumes its search for a condition handler, using the search order described above. If it finds another condition handler, it passes control to that routine.

3. Unwind

The condition handler cannot fix the problem, and execution cannot continue using the current flow. The handler issues the Unwind Call Stack ($UNWIND) system service to unwind the call stack. Call frames may then be removed from the stack and the flow of execution modified, depending on the arguments to the $UNWIND service.

Examples of these three situations are shown in the following sections.
Table 3-5  
Summary of Exception Conditions

<table>
<thead>
<tr>
<th>Condition Name/Type</th>
<th>Explanation</th>
<th>Additional Arguments</th>
</tr>
</thead>
</table>
| SSS\_ACCVID         | Access violation | 1. Reason for access violation. This is a mask with the format:  
| (Fault)             |             | Bit 0 = type of access violation  
|                     |             | 0 = page table entry protection code did not permit intended access  
|                     |             | 1 = PSLR, PIR, or SILR length violation  
|                     |             | Bit 1 = page table entry reference  
|                     |             | 0 = specified virtual address not accessible  
|                     |             | 1 = associated page table entry not accessible  
|                     |             | Bit 2 = intended access  
|                     |             | 0 = read  
|                     |             | 1 = modify  
|                     |             | 2. Virtual address to which access was attempted |
| SSS\_ARTRES         | Reserved arithmetic trap | None |

| SSS\_ASTFLT         | Stack invalid during attempt to deliver an AST | 1. Stack pointer value when fault occurred  
| (Fault)             |             | 2. AST parameter of failed AST  
|                     |             | 3. Program counter (PC) at AST delivery interrupt  
|                     |             | 4. Processor status longword (PSL) at AST delivery interrupt  
|                     |             | 5. Program counter (PC) to which AST would have been delivered  
|                     |             | 6. Processor status longword (PSL) to which AST would have been delivered  |
| SSS\_BREAK          | Breakpoint instruction encountered | None. |

| SSS\_CMODSUPR       | Change mode to supervisor instruction encountered | Change mode code. The possible values are -32768 through 32767. |
| (Trap)              |             | |

| SSS\_CMODUSER       | Change mode to user instruction encountered | Change mode code. The possible values are -32768 through 32767. |
| (Trap)              |             | |

1 The PC and PSL normally included in the signal array are not included in this argument list. The stack pointer of the access mode receiving this exception is reset to its initial value.

2 If a change mode handler has been declared for user or supervisor modes with the Declare Change Mode or Compatibility Mode Handler (SDCLCMH) system service, that routine receives control when the associated trap occurs.
# How to Use System Services

## Condition Handling Services

<table>
<thead>
<tr>
<th>Condition Name/Type</th>
<th>Explanation</th>
<th>Additional Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>SS$_COMPAT (Fault)</td>
<td>Compatibility mode exception. This exception condition can only occur when executing in 3 compatibility mode.</td>
<td>Type of compatibility exception. The possible values are: 0 = Reserved instruction execution, 1 = BPT instruction executed, 2 = IOT instruction executed, 3 = EMT instruction executed, 4 = TRAP instruction executed, 5 = Illegal instruction executed, 6 = Odd address fault, 7 = TBIT trap</td>
</tr>
<tr>
<td>SS$_DECOVF ( Trap)</td>
<td>Decimal overflow</td>
<td>None</td>
</tr>
<tr>
<td>SS$_FLTDIV (Trap)</td>
<td>Floating/decimal divide by zero</td>
<td>None</td>
</tr>
<tr>
<td>SS$_FLTOVF (Trap)</td>
<td>Floating overflow</td>
<td>None</td>
</tr>
<tr>
<td>SS$_FLTUND (Trap)</td>
<td>Floating underflow</td>
<td>None</td>
</tr>
<tr>
<td>SS$_INTDIV (Trap)</td>
<td>Integer divide by zero</td>
<td>None</td>
</tr>
<tr>
<td>SS$_INTOVP (Trap)</td>
<td>Integer overflow</td>
<td>None</td>
</tr>
<tr>
<td>SS$_OPCCUS (Fault)</td>
<td>Opcode reserved to customer fault</td>
<td>None</td>
</tr>
<tr>
<td>SS$_OPCDEC (Fault)</td>
<td>Opcode reserved to Digital fault</td>
<td>None</td>
</tr>
<tr>
<td>SS$_PAGRDERR (Fault)</td>
<td>Read error occurred during an attempt to read a faulted page from disk</td>
<td>1. Translation not valid reason. This is a mask with the format: Bit 0 = 0 Bit 1 = page table entry reference 0 = specified virtual address not valid 1 = associated page table entry not valid Bit 2 = intended access 0 = read 1 = modify</td>
</tr>
<tr>
<td>SS$_RADRMOD (Fault)</td>
<td>Attempt to use a reserved addressing mode</td>
<td>None</td>
</tr>
<tr>
<td>SS$_ROPRAND (Fault)</td>
<td>Attempt to use a reserved operand</td>
<td>None</td>
</tr>
<tr>
<td>SS$_SSFFAIL (Fault)</td>
<td>System service failure (when system service failure exception mode is enabled)</td>
<td>Status return from system service (RU) (The same value is in RU of the mechanism array)</td>
</tr>
<tr>
<td>SS$_SUBRNG</td>
<td>Subscript range trap</td>
<td>None</td>
</tr>
<tr>
<td>SS$_TBIT (Fault)</td>
<td>Trace bit is pending following an instruction</td>
<td>None</td>
</tr>
</tbody>
</table>

3 If a compatibility mode handler has been declared with the Declare Change Mode or Compatibility Mode Handler ($DCLCMH) system service, that routine receives control when this fault occurs.
3.7.6 Example of Condition Handling Routines Continuing and Resignaling

Figure 3-20 shows two procedures, A and B, that have declared condition handlers. The notes describe the sequence of events that would occur if a call to a system service failed during the execution of Procedure B.

```
PGMA:: .WORD 0
 MOVAL HANDLER A,(FP) ①
 $SET S F M _ S E N B F L G = # 1
 CALLG ARGLIST, PGMB ②

HANDLER A:
 .WORD "M< R2>
 MOVL CHF$L.SIGARGLST(AP), R4 ⑥
 CMPL $SS$.SSFAIL, CHF$L.SIG_NAME(R4)
 BNEQ 10$
 ⑤
 MOVZWL $SS$.CONTINUE,R0
 RET ④

10$:
 MOVZWL $SS$.RESIGNAL,R0
 RET

PGMB:: .WORD "M< R2,R3,R4>
 MOVAL HANDLER B,(FP) ③

 ④ System service failure occurs

HANDLER B:
 .WORD "M< R2,R3,R4>
 MOVL CHF$L.SIGARGLST(AP), R4
 CMPL $SS$.BREAK, CHF$L.SIG_NAME(R4)
 BNEQ 10$
 ⑤
 MOVZWL $SS$.CONTINUE,R0
 RET

10$:
 MOVZWL $SS$.RESIGNAL,R0 ⑥
 RET
```
Procedure A executes and establishes condition handler HANDLER A. HANDLER A is set up to respond to exceptions caused by failures in system service calls.

During its execution, Procedure A calls Procedure B.

Procedure B establishes condition handler HANDLER B. HANDLER B is set up to respond to breakpoint faults.

While Procedure B is executing, an exception condition occurs caused by a system service failure.

The exception dispatcher searches the exception vectors for a condition handler (assume there are none defined), and then searches the call stack. HANDLER B is called with the condition SS$_SSFFAIL.

Since HANDLER B only handles breakpoint faults, it places the return value SS$_RESIGNAL in R0 and returns control to the exception dispatcher.

The exception dispatcher resumes its search for a condition handler and calls HANDLER A.

HANDLER A handles the system service failure exception, corrects the condition, and the return value SS$_CONTINUE in R0, and returns control to the exception dispatcher.

The dispatcher returns control to Procedure B, and execution of Procedure B resumes at the instruction following the system service failure.
3.7.7 Unwinding the Call Stack

The third course of action a condition handler can take is to unwind the procedure call stack. The unwind operation is complex, and should only be used when control must be restored to an earlier procedure in the calling sequence. Moreover, use of the SUNWIND system service requires the calling condition handler to be aware of the calling sequence and of the exact point to which control is to return.

The SUNWIND system service accepts two optional arguments:

1. The depth to which the unwind is to occur. If the depth is 1, the call stack is unwound to the caller of the procedure that caused the exception condition. If the depth is 2, the unwind is to the caller's caller, and so on.

2. The address of a location to receive control when the unwind is complete, that is, a return PC to replace the current PC in the call frame of the procedure that will receive control when all specified frames have been removed from the stack.

If no arguments are supplied to the SUNWIND service, the unwind is performed to the caller of the procedure that established the condition handler that is issuing the SUNWIND service. Control is returned to the address specified in the return PC for that procedure. Note that this is the default and normal case for unwinding.

Figure 3-21 illustrates an unwind situation and describes some of the possible results.

During the actual unwinding of the call stack, the unwind routine examines each frame in the call stack to see if a condition handler has been declared. If a handler has been declared, the unwind routine calls the handler with the code SSS UNWIND in the condition name argument of the signal array. When a condition handler is called with this condition, it can perform any procedure-specific cleanup operations required. After the handler returns, the call frame is removed from the stack.

Thus, in Figure 3-21, HANDLERB may be called a second time, during the unwind operation. Note that HANDLERB does not have to be able to specifically interpret the SSS UNWIND condition; the RET instruction merely returns control to the unwind procedure, which does not check any status values.
HOW TO USE SYSTEM SERVICES
CONDITION HANDLING SERVICES

Notes:

1. The procedure call stack is as shown. Assume that no exception vectors are declared for the process and that the exception condition occurs during the execution of Procedure D.

2. Since neither Procedure D nor Procedure C has established a condition handler, HANDLERB receives control.

3. If HANDLERB issues the $UNWIND system service with no arguments, the call frames for B, C, and D are removed from the stack (along with the call frame for HANDLERB itself), and control returns to Procedure A. Procedure A receives control at the point following its call to Procedure B.

4. If HANDLERB issues the $UNWIND system service specifying a depth of 2, call frames for C and D are removed, and control returns to Procedure B.

Figure 3-21 Unwinding the Call Stack
3.7.8 Multiple Exception Conditions

It is possible for a second exception condition to occur while a condition handler or a procedure that it has called is still executing. In this case, when the exception dispatcher searches for a condition handler, it skips the frames that were searched to locate the first handler.

The search for a second handler terminates in the same manner as the initial search, as described in Section 3.7.3.

If the $UNWIND system service is issued by the second active condition handler, the depth of the unwind is determined according to the same rules followed in the exception dispatcher's search of the stack: all frames that were searched for the first condition handler are skipped.

If an exception occurs during the execution of a handler established in the primary or secondary exception vector, that handler must handle the additional condition.
3.8 MEMORY MANAGEMENT SERVICES

The VAX/VMS memory management routines map and control the relationship between physical memory and a process's virtual address space. These activities are, for the most part, transparent to you, as a user, and to your programs. However, you can in some cases, make a program more efficient by explicitly controlling its virtual memory usage. Memory management services allow you to:

- Increase or decrease the virtual address space available in a process's program or control region
- Control the process's working set size and the swapping of pages between physical memory and the paging device
- Define disk files containing data or shareable images and map the file into the process's virtual address space

This section discusses the services that provide these capabilities. However, before you use any of these services, you should have an understanding of the VAX-11 memory structure and memory management routines. Where pertinent, virtual memory concepts related to the use of particular services are discussed in this section. For more background information, see the VAX/VMS Summary Description.

3.8.1 Increasing Virtual Address Space

The virtual address space of a process is divided into two regions:

1. The program (P0) region contains the image currently being executed.

2. The control (P1) region contains the information maintained by the system on behalf of the process. It also contains the user stack, which is located at the lower-addressed end of the control region.

Figure 3-22 illustrates the layout of a process's virtual memory. The initial size of a process's virtual address space depends on the size of the image being executed.

To facilitate memory protection and mapping, the virtual address space is subdivided into 512-byte units called pages. Using memory management services, a process can add a specified number of pages to the end of either the program region or the control region. Adding pages to the program region provides the process with additional space for image execution; for example, for the dynamic creation of tables or data areas. Adding pages to the control region increases the size of the user stack. (The user stack can also be expanded when the image is linked.)

The maximum size to which a process can increase its address space is controlled by an entry in the system authorization file for the user.
3.8.2 Increasing and Decreasing Virtual Address Space

The Expand Program/Control Region ($EXPREG) system service adds pages to the end of either the program or control region, and optionally returns the range of virtual addresses of the new pages. For example, if you want to add four pages to a process's program region, you can code a call to the $EXPREG system service as follows:

```
BEGSPACE:
  .BLKL 2
  #RETURN START AND END OF NEW PAGES
  ;
  $EXPREG_S PAGCNT=$4, RETADR=BEGSPACE, REGION=$0; GET 4 PAGES
```

To add the same number of pages to the control region, you would specify REGION=$1.

When pages that have been added at the end of a region are no longer needed, they can be deleted with the Contract Program/Control Region ($CNTREG) system service. As for the $EXPREG service, you code the number of pages you want deleted and the region:

```
$CNTREG_S PAGCNT=$4, REGION=$0
```

Note that the REGION argument for both the $EXPREG and $CNTREG services is optional; if not specified, the pages are added to or deleted from the program region, by default.
The $EXPREG and $CNTREG services can only add or delete pages from the end of a particular region. When you need to add or delete pages that are not at the end of these regions, you can use the Create Virtual Address Space ($CRETVA) and Delete Virtual Address Space ($DELTVA) system services. For example, if you have used the $EXPREG service twice to add pages to the program region, and want to delete the first range of pages, but not the second, you could use the $DELTVA system service as shown in the following sequence:

```
BEGSPACEA: .BLKL 2 ;START AND END OF FIRST AREA
BEGSPACEB: .BLKL 2 ;START AND END OF SECOND AREA

$EXPREG_S PAGCNT=#4,RETADR=BEGSPACEA,REGION=#0 ;FOUR PAGES
BSBW ERROR

$EXPREG_S PAGCNT=#3,RETADR=BEGSPACEB,REGION=#0 ;THREE PAGES
BSBW ERROR

$DELTVA_S INADR=BEGSPACEA ;DELETE FIRST 4 PAGES
BSBW ERROR
```

In the above example, the first call to $EXPREG adds four pages to the program region; the virtual addresses of the pages are returned in the 2-longword array at BEGSPACEA. The second call adds three pages, and returns the addresses at BEGSPACEB. The call to $DELTVA deletes the first four pages that were added.

3.8.2.1 Input Address Arrays and Return Address Arrays - When the $EXPREG system service adds pages to a region, it adds them in the normal direction of growth for the region. The return address array, if requested, indicates the order in which the pages were added:

- If the program region is expanded, the starting virtual address is lower than the ending virtual address.
- If the control region is expanded, the starting virtual address is higher than the ending virtual address.

Conversely, the direction of contraction with the $CNTREG system service is from a higher to a lower address in the program region and from a lower to a higher address in the control region.

The addresses returned indicate the first byte in the first page added or deleted and the last byte in the last page added or deleted.

When input address arrays are specified for the Create or Delete Virtual Address Space system services ($CRETVA and $DELTVA, respectively), these services add or delete pages beginning with the address specified in the first longword and ending with the address specified in the second longword.
HOW TO USE SYSTEM SERVICES
MEMORY MANAGEMENT SERVICES

The order in which the pages are added or deleted does not have to be in the normal direction of growth for the region. Moreover, since these services only add or delete whole pages, they ignore the low-order 9 bits of the specified virtual address (the low-order 9 bits contain the byte address). The virtual addresses returned do indicate the byte addresses.

Table 3-6 shows some sample virtual addresses that might be specified as input to $CRETVA or $DELTVA and shows the return address arrays, if all pages are successfully added or deleted.

<table>
<thead>
<tr>
<th>Input Array Start</th>
<th>Input Array End</th>
<th>Region</th>
<th>Output Array Start</th>
<th>Output Array End</th>
<th>Number of Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>1010</td>
<td>1670</td>
<td>P0</td>
<td>1000</td>
<td>17FF</td>
<td>6</td>
</tr>
<tr>
<td>1450</td>
<td>1451</td>
<td>P0</td>
<td>1400</td>
<td>15FF</td>
<td>1</td>
</tr>
<tr>
<td>1450</td>
<td>1450</td>
<td>P0</td>
<td>1400</td>
<td>15FF</td>
<td>1</td>
</tr>
<tr>
<td>7FPEC010</td>
<td>7FPEC010</td>
<td>P1</td>
<td>7FPEC00FF</td>
<td>7FPEC000</td>
<td>1</td>
</tr>
</tbody>
</table>

Note that if the input virtual addresses are the same, a single page is added or deleted. The return address array indicates that the page was added or deleted in the normal direction of growth for the region.

3.8.3 Page Ownership and Page Protection

Each page in a process's virtual address space is owned by a particular access mode. The owner is the access mode that created the page. For example, pages in the program region initially provided for the execution of an image are owned by user mode. Pages that the image creates dynamically are also owned by user mode. Pages in the control region, except for the pages containing the user stack, are normally owned by more privileged access modes.

Only the owner of a page can delete the page or otherwise affect it. The owner of a page can also indicate, by means of a protection code, the type of access that each access mode will be allowed.

The Set Protection on Pages ($SETPRT) system service changes the protection assigned to a page or group of pages. The protection is expressed as a code that indicates the specific type of access (none, read-only, read, or write) for each of the four access modes (kernel, executive, supervisor, user). Only the owner access mode or a more privileged access mode can change the protection for a page.

When an image attempts to access a page that is protected against the access attempted, a hardware exception, called an access violation, occurs. When an image calls a system service, the service determines whether an access violation would occur when the image attempted to read or write a page it is not privileged to access. If so, the service returns the status code SS$ACCVIO.
HOW TO USE SYSTEM SERVICES
MEMORY MANAGEMENT SERVICES

Since the memory management services add, delete, or modify a single page at a time, one or more pages can be successfully affected before an access violation is detected. If the RETADR argument is specified in the service call, the service returns the addresses of pages actually affected before the error. If no pages are affected, that is, if an access violation would occur on the first page specified, the service returns a -1 in both longwords of the return address array.

If the RETADR argument is not specified, no information is returned.

3.8.4 Working Set Paging

When a process is executing an image, a subset of its pages resides in physical memory; these pages are called the process’s working set. The working set includes pages in both the program region and the control region.

When the image refers to a page that is not in memory, a hardware fault occurs, and the page is brought into memory, replacing an existing page in the working set. If the page that is going to be replaced has been modified during the execution of the image, that page is written onto a secondary storage device, called the paging device. When this page is needed again, it is brought back into memory, again replacing a current page from the working set. This exchange of pages between physical memory and secondary storage is called paging.

The paging of a process's working set is transparent to the process. However, if a program is very large, or if pages in the program image that are heavily used are being paged in and out frequently, the overhead required for paging may decrease the program's efficiency. Some system services allow a process, within limits, to counteract these potential problems:

- The Adjust Working Set Limit ($ADJWSL) system service increases the maximum number of pages that a process can have in its working set.
- The Purge Working Set ($PURGWS) system service removes page from the working set.
- The Lock Pages in Working Set ($LKWSL) system service makes a page or pages in the working set ineligible for paging.

The initial size of a process's working set is defined by the process's working set default (WSDEFAULT) quota. Since some programs may have larger memory requirements than others, a program can call the $ADJWSL system service to dynamically increase the process's working set limit. When the additional pages are no longer needed in the working set, the program can call the $ADJWSL service to decrease the working set limit. Or, it can call the $PURGWS system service to remove pages no longer in use from the working set.

When the system pages a process's working set, the pages in the working set are paged on a first-in, first-out basis. Under some circumstances, an image may not want certain pages to be paged out at all; then, it can lock them in the working set. As long as the process's working set is in memory, these pages cannot be paged out until they are explicitly unlocked with the Unlock Pages in Working Set ($ULWSL) system service.

3-81
3.8.5 Process Swapping

The operating system balances the needs of all the processes that are currently executing, providing each with the system resources it requires on an as-needed basis. The memory management routines balance the process's memory requirements. Thus, the sum of the working sets for all processes that are currently in physical memory is called the balance set.

When a process whose working set is in memory becomes inactive -- for example to wait for an I/O request or to hibernate -- the entire working set may be removed from memory to provide space for another process's working set to be brought in for execution. This removal of a process's working set is called swapping. When a process is swapped out of the balance set, all of the pages of its working set (modified and unmodified pages) are swapped, including any pages that were locked in the working set.

It is possible for a high-priority process to lock its entire working set in the balance set. While pages can still be paged in and out of the working set, the process remains in memory even when it is inactive. To lock itself in the balance set, the process issues the Set Process Swap Mode ($SETSWM) system service. For example:

$SETSWM_S SWPFLG=$1

This call to $SETSWM disables process swap mode. Swap mode can also be disabled by setting the appropriate bit in the STSFLG argument to the Create Process ($CREPRC) system service. A user privilege is required, however, to alter process swap mode.

Another way that a process can lock pages in memory is with the Lock Pages in Memory ($LCKPAG) system service. When a page is locked in memory with this service, the page remains in memory even when the remainder of the process's working set is swapped out of the balance set. This system service has limited applicability, but may be useful in special circumstances, for example, for routines that perform I/O operations to slow devices or graphics devices.

Pages locked in memory can be unlocked with the Unlock Pages in Memory ($ULKPAG) system service. The user privilege $PSWAPM is required to issue both of these services.

3.8.6 Sections

Sections are disk files or portions of disk files containing data or code that can be brought into memory and made available to a process for manipulation and execution. Sections are either private or shared:

- Private sections are accessible only by the process that creates them; a process can define a disk data file as a section, map it into its virtual address space, and manipulate it.

- Global sections can be shared by more than one process. One copy of the global section resides in physical memory, and each process sharing it refers to the same copy. A global section can contain shareable code or data that can be read, or read and written, by more than one process. Global sections are either temporary or permanent, and can be defined for use within a group or on a system-wide basis.
HOW TO USE SYSTEM SERVICES
MEMORY MANAGEMENT SERVICES

When pages in sections are paged out of memory during image execution, they are written back into the section file, rather than onto secondary storage, as is the normal case.

The use of sections involves two distinct operations:

1. The creation of a section defines a disk file as a section and informs the system what portions of the file contain the section.

2. The mapping of a section makes the section available to a process and establishes the correspondence between virtual blocks in the file and specific addresses in the process's virtual address space.

The Create and Map Section ($CRMPSC) system service creates and/or maps a private section or a global section. Since a private section is used only by a single process, creation and mapping are simultaneous operations. In the case of a global section, one process can create a permanent global section and not map it; other processes can map to it. A process can also create and map a global section in one operation.

The following sections describe creating, mapping, and using sections. In each case, considerations that are common to both private sections and global sections are described first, followed by additional notes and requirements for the use of global sections.

3.8.6.1 Creating Sections - The steps involved in section creation are:

1. Opening or creating the disk file containing the section

2. Defining which virtual blocks in the file comprise the section

3. Defining the characteristics of the section

3.8.6.2 Opening the Disk File - Before a file can be used as a section, it must be opened using RMS.

The following example shows the file access block (FAB), OPEN macro, and channel specification on the $CRMPSC system service to open an existing file for reading:

SECFA: $FAB  FNM=<SECTION,TST>,FOP=UFO  FILE ACCESS BLOCK

$OPEN  FAB=SECFA
$CRMPSC_S CHAN=SECFA+FAB+L_STV,...

The file options (FOP) parameter indicates that the file is to be opened for user I/O; this option is required so that RMS assigns the channel using the access mode of the caller. RMS returns the channel number on which the file is accessed in the offset FAB$L_STV; this channel number is specified as input to the $CRMPSC system service (CHAN argument). The same channel number can be used for multiple create and map section operations. It can also be used to read and write virtual blocks to the section file with the Queue I/O Request ($SQIO) system service.

3-83
The file may be a new file that is to be created while it is in use as a section. In this case, use the $CREATE macro to open the file. If you are creating a new file, the file access block (FAB) for the file must specify an allocation quantity (ALQ parameter).

$CREATE can also be used to open an existing file; if the file does not exist, it will be created. The following example shows the required fields in the FAB for the conditional creation of a file:

```
GBLFAB: $FAB    FNM=<GLOBAL.TST>,ALQ=4,FAC=PUT,-
            FDF=<UFO,CIF,CBT>
```

When the $CREATE macro is invoked, it creates the file GLOBAL.TST if the file does not currently exist. The CBT (contiguous-best-try) option requests that if possible, the file be contiguous. Although it is not required that section files be contiguous, better performance can result if they are.

3.8.6.3 Defining the Section Extents - Once the file is successfully opened, the $CRMPSC system service can create a section from the entire file, or from only certain portions of it. The following arguments to $CRMPSC define the extents of the file that comprise the section:

- **PAGCNT (page count)**. This argument is required; it indicates the number of virtual blocks in the file. These blocks correspond to pages in the section.

- **VBN (virtual block number)**. This argument defines the number of the virtual block in the file that is the beginning of the section. It is an optional argument; if not specified, it defaults to 1; that is, the first virtual block in the file is the beginning of the section.

3.8.6.4 Defining the Section Characteristics - The FLAGS argument to the $CRMPSC system service defines the following section characteristics:

- Whether it is a private section or a global section (the default is to create a private section)

- How the pages of the section are to be treated when they are copied into physical memory or when a process refers to them. The pages in a section can be:
  - --read/write or read-only
  - --created as demand-zero pages or as copy-on-reference pages, depending on how the processes are going to use the section and whether the file contains any data (see Section 3.8.6.8, "Section Paging").
3.8.6.5 Defining Global Section Characteristics - If the section is a
global section, it must be assigned a character string name (GSDNAM
argument) so that other processes can identify it when they are
mapping it.

The FLAGS argument specifies the type of global section:

- Group temporary (the default)
- Group permanent
- System temporary
- System permanent

Group global sections can be shared only by processes executing with
the same group number. The name of a group global section is
implicitly qualified by the group number of the process that created
it. When other processes map to it, their group numbers must match.

A temporary global section is automatically deleted when no processes
are mapped to it.

Permanent global sections remain in existence even when no processes
mapped to them. They must be explicitly marked for deletion with the
Delete Global Section ($DGBLSC) system service.

The user privileges PRMGBL and SYSGBL are required to create permanent
group global sections, or system global sections (temporary or
permanent), respectively.

A system global section can be made available to all processes in the
system.

Optionally, a process creating a global section can specify a file
protection mask (PROT argument), restricting all access or a type of
accccc (read, write, extend, delete) to other processes.

3.8.6.6 Mapping Sections - When you code the $CRMPSC system service
to create and/or map a section, you must provide the service with a
range of virtual addresses (INADR argument) into which the section is
to be mapped.

If you know specifically which pages the section should be mapped
into, you provide these addresses in a 2-longword array. For example,
to map a private section of 10 pages into virtual pages 10 through 19
of the program region, specify the input address array as follows:

MAPRANGE:

.LONG ^X1400  ADDRESS (HEX) OF PAGE 10
.LONG ^X2300  ADDRESS (HEX) OF PAGE 19

The addresses specified do not have to be currently in the process's
virtual address space. The $CRMPSC system service calls the Create
Virtual Address Space ($CREVA) system service to create the required
virtual address space before mapping the section. If you code the
RETADR argument, the service returns the range of addresses actually
mapped.

You do not need to know explicit addresses to provide an input address
range. If you want the section mapped into the current end of the
program region, you can use the $EXPREG system service to add the
pages at the end of the program region and use the return address
array from $EXPREG as input to the $CRMPSC system service.
HOW TO USE SYSTEM SERVICES
MEMORY MANAGEMENT SERVICES

You can also obtain the address of the next available page in the region by calling the Get Job/Process Information ($GETJPI) system service. The $GETJPI service returns an address you can use for the starting address in the range. You then provide a very high address in the program region as the ending address: $CRMPSC creates only as many pages as necessary to map the section, and returns the addresses mapped in the return address array. The following example shows such a sequence:

GETVADR: .WORD 4 ;LENGTH OF BUFFER
         .WORD JPI$_FREP0VA ;ITEM IDENTIFIER
         .LONG MAPRANGE ;ADDRESS OF BUFFER
         .LONG 0 ;NOT NEEDED
         .LONG 0 ;END OF JPI LIST
MAPRANGE:
         .BLKL 1
         .LONG ^X1FFFFFFF ;VERY LARGE ADDRESS
RETRANGE:
         .BLKL 2

*GETJPI_S ITMLST=GETVADR ;FIND FIRST FREE PAGE
*CRMPSC_S INADR=MAPRANGE,RETAISR=RETRANGE,...

The item code JPI$_FREP0VA is defined in the $JPIDEF macro. For complete details on how to use the $GETJPI system service, see the service description in Chapter 4.

Once a section has been successfully mapped, the image can refer to the pages using a base register and predefined symbolic offset names or labels defining offsets of an absolute program section or structure.

Figure 3-23 shows an example of creating and mapping a process section.

SECFAB: *FAB     FNML=<SECTION.TST>,FOP=UFO,FAC=PUT

MAPRANGE:
         .LONG ^X1400 ;FIRST PAGE
         .LONG ^X2300 ;LAST PAGE
RETRANGE:
         .BLKL 1 ;FIRST PAGE MAPPED
ENDRANGE:
         .BLKL 1 ;LAST PAGE MAPPED

1 *OPEN FAB=SECFAB ;OPEN SECTION FILE
     SBW ERROR

2 *CRMPSC_S INADR=MAPRANGE,- ;INPUT ADDRESS ARRAY
   RETADR=RETRANGE,- ;OUTPUT ARRAY
   PAGCNT=4,- ;MAP FOUR PAGES
3 FLAGS=SEC$M_WRTE ;READ/WRTE SECTION
   CHAN=SECFAB$FAB$4_LSTV ;CHANNEL NUMBER
     SBW ERROR

4 MOV R RETRANGE,R6 ;POINT TO START OF SECTION

Figure 3-23 Creating and Mapping a Private Section
HOW TO USE SYSTEM SERVICES
MEMORY MANAGEMENT SERVICES

Notes on Figure 3-23:

1. The OPEN macro opens the section file defined in the file access block SEC Fab.

2. The SCRMPSC system services uses the addresses specified at MAPRANGE to specify an input range of addresses into which the section will be mapped. The PAGCNT argument requests that only four pages of the file be mapped.

3. The FLAGS argument requests that the pages in the section be read/write. The symbolic flag definitions for this argument are defined in the $SECDEF macro. Note that the file access field (FAC parameter) in the FAB also indicates that the file is to be opened for writing.

4. When SCRMPSC completes, the addresses of the four pages that were mapped are returned in the output address array at RETRANGE. The address of the beginning of the section is placed in register 6, which serves as a pointer to the section.

3.8.6.7 Mapping Global Sections - A process that creates a global section can map to it when it creates it. Then, other processes can map it by calling the Map Global Section ($MGBLSC) system service.

When a process maps a global section, it must specify the global section name assigned to the section when it was created, whether it is a group or system global section, and whether it desires read-only or read/write access. The process may also specify:

- A version identification (IDENT argument), indicating the version number of the global section (when multiple versions exist) and whether more recent versions are acceptable to the process.

- A relative page number (RELPAG argument), specifying the page number, relative to the beginning of the section, to begin mapping the section. In this way, processes can use only portions of a section. Additionally, a process can map a piece of a section into a particular address range and subsequently map a different piece of the section into the same virtual addresses.

Cooperating processes can both issue a SCRMPSC system service to create and map the same global section. The first process to call the service actually creates the global section; subsequent attempts to create and map the section result only in mapping the section for the caller. The successful return status code $S$ CREATED indicates that the section did not already exist when the SCRMPSC system service was called. If the section did exist, the status code $S$$_$NORMAL is returned.

Figure 3-24 shows an example of the creation of a global section, and a second process mapping the section.
HOW TO USE SYSTEM SERVICES
MEMORY MANAGEMENT SERVICES

Process ORION

GBLCLUSTER:

;COMMON EVENT FLAG CLUSTER NAME
DESRIPTOR <GLOBAL_CLUuster>

GBLSET = 65 ;FLAG NUMBER TO ASSOCIATE AND SET
GBLWAIT = 66 ;FLAG NUMBER TO WAIT FOR

GLOBALSEC:

;GLOBAL SECTION NAME
DESRIPTOR <GLOBAL SECTION>

GBLFAG: $FAB
FN=$GLOBAL_TST>,FOP=$UFO,CIF,CRF>,ALQ=4,FAC=PUT

;ASCEFC.S EFN=#GBLSET,NAME=GBLCLUSTER
BSBW ERROR

;ASCEFC_S EFN=#GBLSET,NAME=GLOBALSEC
BSBW ERROR

;CREATE GLOBAL SECTION
FLAGS=#SEC$M_WRIT!SEC$M_GBL,...

;SET COMMON EVENT FLAG

Process CYGNUS

CLUSTER: DESCRIPTOR <GLOBAL_CLUuster>;CLUSTER NAME

GBLSET = 65
GBLWAIT = 66

SECTION: DESCRIPTOR <GLOBAL SECTION>;SECTION NAME

;ASCEFC.S EFN=#GBLSET,NAME=CLUSTER
BSBW ERROR

;WAITFR.S EFN=#GBLSET
BSBW ERROR

;MGBLSCL S INAD=MAPRANGE,RETADR=RETRANGE,
FLAGS=#SEC$M_GBL,...;GLOBAL SECTION
GSDNAM=SECTION;SECTION NAME
BSBW ERROR

Figure 3-24 Creating and Mapping a Global Section

Notes on Figure 3-24:

1. The processes ORION and CYGNUS are in the same group. Each process first associates with a common event flag cluster named GLOBAL CLUSTER to use common event flags to synchronize their use of the section.

2. ORION creates the global section named GLOBAL SECTION, specifying flags that indicate that it is a global section (SEC$M_GBL) and that it is read/write. Input and output address arrays, the page count parameter and the channel number arguments are not shown; procedures for coding them are the same as shown in Figure 3-23.
The process CYGNUS associates with the common event flag cluster and waits for the flag defined as GBLSET. ORION sets this flag when it has completed creating the section. To map the section, CYGNUS specifies the input and output address arrays, the flag indicating that it is a global section, and the global section name. The number of pages mapped is always the same as that specified by the creator of the section.

3.8.6.8 Section Paging - The first time that an image executing in a process refers to a page that was created during the mapping of a section, the page is copied into physical memory. The address of the page in the process's virtual address space is mapped to the physical page. During the execution of the image, normal paging can occur; however, pages in sections are not written onto secondary storage devices when they are paged out, as is the normal case. Rather, if they have been modified, they are written back into the section file on disk. The next time a page fault occurs for the page, the page is brought back from the section file.

In the case of global sections, more than one process can be mapped to the same physical pages. These pages are paged out, and written back to the disk file defined as the section, only when no processes are currently mapped to them.

If the pages in a section are defined as demand-zero pages or copy-on-reference pages when the section was created, the pages are treated differently.

If the call to $CRMPSC requested that pages in the section be treated as demand-zero pages, these pages are initialized to zeros when they are first brought into physical memory. If the file is either a new file that is being created as a section or a file that is being completely rewritten, demand-zero pages provide a convenient way of initializing the pages.

If the call to $CRMPSC requested that pages in the section be copy-on-reference pages, each process that maps to the section receives its own copy of the section, on a page-by-page basis from the file, as it refers to them. These pages are never written back into the section file.

3.8.6.9 Reading and Writing Data Sections - Read/write sections provide a way for a process, or cooperating processes, to manipulate data files in virtual memory.

The sharing of global sections may involve application-dependent synchronization techniques. For example, one process can create and map to a global section in read/write status; other processes can map to it in read-only status, and interpret data written by the first process. Or, two or more processes can write to the section concurrently. (In this case, the application program must provide the necessary synchronization and protection.)

When a file that has been mapped as a section is written back to disk, its version number is not incremented but the revision number is. A full directory listing indicates the revision number of the file and the date and time that the file was last updated.
When the file has been updated, the process or processes can release, or unmap, the section. The section is then written back into the disk file defined as a section.

3.8.6.10 Releasing and Deleting Sections - A process unmaps a section by deleting the virtual addresses in its own virtual address space to which it has mapped the section. If a return address range was specified to receive the virtual addresses of the mapped pages, this address range can be used as input to the Delete Virtual Address Space ($DELTVA) system service. For example:

`$DELTVA_S INADR=RET_RANGE`

When a process unmaps a private section, the section is deleted; that is, all control information maintained by the system is deleted. A temporary global section is deleted when all processes that have mapped to it have unmapped it. Permanent global sections are not deleted until they are specifically marked for deletion with the Delete Global Section ($DGBLSG) system service; then, they are deleted when no more processes are mapped.

Note that deleting the pages occupied by a section does not delete the section file, but rather cancels the process's association with the file. Moreover, when a process deletes pages mapped to a read/write section and no other processes are mapped to it, all modified pages are written back into the section file.

When all processes mapped to a section have deleted the pages into which the section was mapped from their virtual address space, the channel can be deassigned. The process that created the section can deassign the channel (with the Deassign I/O Channel system service), for example:

`$DASSGN_S CHAN=GBLFAB+FAB$L_STV`

3.8.6.11 Checkpointing Sections - Since read/write sections are normally not updated on disk until the physical pages they occupy are paged out, or until all processes referring to the section have unmapped it, a process may have to ensure that all modified pages have been successfully written back into the section file.

The Update Section File on Disk ($UPDSEC) system service writes the modified pages in a section into the disk file. The $UPDSEC system service is described in Chapter 4.

3.8.6.12 Image Sections - Global sections can contain shareable code. An image file that is going to be defined as a section must contain position-independent code.

The operating system uses global sections to implement shareable code as follows:

1. The object module containing code to be shared is linked to produce a shareable image. The shareable image is not, in itself, executable. It contains a series of sections, called image sections.
2. A user links private object modules with the shareable image to produce an executable image. Only image section descriptor records from the shareable image file are bound with the image sections from the user's code.

3. The system manager uses the INSTALL command to create a permanent global section from the shareable image file making the image sections available for sharing.

4. When the user runs the executable image, the system automatically maps the global sections created by the INSTALL command into the virtual address space of the user's process.

For details on how to create and identify shareable images, and how to link them with private object modules, see the VAX-11 Linker Reference Manual. For information on installing shareable images and making them available for sharing as global sections, see the VAX/VMS System Manager's Guide.
CHAPTER 4

SYSTEM SERVICE DESCRIPTIONS

This chapter describes each of the VAX/VMS system services. The services are presented in alphabetical order, by their abbreviated names.

Each system service description consists of the following categories, as applicable:

Macro Format:

Shows the macro name, with all keyword arguments listed in positional order. Spaces between arguments are present for readability, and are not part of the macro syntax.

High-Level Language Format:

Shows the procedure name and a generalized format for calling the service from a high-level language, with all arguments listed in positional order. Spaces between arguments are present for readability, and are not part of the statement syntax.

Arguments...

Describes each of the arguments.

Return Status:

Lists the possible return status codes from the service with an explanation of the return condition. The successful returns are listed first, in alphabetical order, followed by warning and severe error return status codes also in alphabetical order. All status codes are severe errors, unless otherwise indicated.

Three severe errors may occur for all services and are not listed with each service description. These are:

SSS_MODE
- The argument list cannot be read by the caller.

SSS_INSARG
- Not enough arguments were supplied to the service.

SSS_ILLSER
- An invalid system service was called.
Privilege Restrictions:

Notes any user privileges required to execute the service or to request a particular function of the service, or any access mode restrictions applied to the service.

Resources Required/Returned:

Lists any system resources or process quotas used by the service, or returned to a process as a result of service execution.

Notes:

Contain the 'fine print' of the service description. All important information pertaining to the service that is not covered in one of the other headings is given here, as well as references to related services or additional information.
4.1 $ADJSTK - ADJUST OUTER MODE STACK POINTER

The Adjust Outer Mode Stack Pointer system service modifies the stack pointer for a less privileged access mode. This service is used by the operating system to modify a stack pointer for a less privileged access mode after placing arguments on the stack.

Macro Format:

$ADJSTK [acmode],[adjust],newadr

High-Level Language Format:

SYSS$ADJSTK([acmode],[adjust],newadr)

acmode
access mode for which the stack pointer is to be adjusted.

adjust
signed adjustment value. The contents of the longword addressed by the NEWADR argument are adjusted by the amount specified in the low-order 16 bits of this argument. The result is loaded into the stack pointer for the specified access mode.

If not specified, or specified as 0, the stack pointer is loaded with the address specified by the NEWADR argument.

newadr
address of a longword to receive the updated value. If the longword contains a nonzero value, then that value is updated by the ADJUST argument value and the result is loaded into the stack pointer.

If the longword contains a 0, the current value of the stack pointer is updated by the ADJUST argument value.

Return Status:

SS$NORMAL
Service successfully completed.

SS$ACCVIO
The longword to store the updated stack pointer or a portion of the new stack segment cannot be written by the caller.

SS$NOPRIV
The specified access mode is equal to or more privileged than the calling access mode.
### SYSTEM SERVICE DESCRIPTIONS

**$ADJSTK - ADJUST OUTER MODE STACK POINTER**

**Note:**

Combinations of zero and nonzero values for the ADJUST argument and the NEWADR longword provide the following results:

<table>
<thead>
<tr>
<th>If the ADJUST argument specifies:</th>
<th>And the longword addressed by NEWADR contains:</th>
<th>The stack pointer is:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>not changed</td>
</tr>
<tr>
<td>0</td>
<td>an address</td>
<td>loaded with the address specified</td>
</tr>
<tr>
<td>a value</td>
<td>0</td>
<td>adjusted by the specified value</td>
</tr>
<tr>
<td>a value</td>
<td>an address</td>
<td>loaded with the specified address, adjusted by the specified value</td>
</tr>
</tbody>
</table>

In all cases, the updated stack pointer value is written into the longword addressed by NEWADR.
4.2 $ADJWSL—ADJUST WORKING SET LIMIT

The Adjust Working Set Limit system service changes the current limit of a process's working set size by a specified number of pages. This service allows a process to control the number of pages resident in physical memory for the execution of the current image.

Macro Format:

$ADJWSL [pagcnt] ,[wsetlm]

High-Level Language Format:

SYS$ADJWSL([pagcnt] ,[wsetlm])

$pagcnt
number of pages to adjust the current maximum working set size.
A positive value increases the maximum working set size; a
negative value decreases it. If not specified, or specified as
0, the current working set size limit is returned in the address
specified by the WSETLM argument, if that argument is coded.

$wsetlm
address of a longword to receive the new working set size limit
or the current working set size limit, if the PAGCNT argument is
not specified.

Return Status:

SS$NORMAL
Service successfully completed.

SS$ACCVIO
The longword to receive the new working set size limit cannot be
written by the caller.

Resources Required/Returned:

The initial value of a process's working set size is controlled
by the working set default quota (WSDEFAULT). The maximum value
to which it may be increased is controlled by the working set
limit quota (WSQUOTA).

Note:

If a program attempts to adjust the working set size beyond the
system-defined upper and lower limits, no error condition is
returned. The working set size is adjusted to the maximum or
minimum size allowed; the caller can check the new working set
size to verify the change.

For more details on memory management concepts and additional services
that help a process control paging and swapping, see Section 3.8,
"Memory Management Services."
SYSTEM SERVICE DESCRIPTIONS

$ALLOC

4.3 $ALLOC - ALLOCATE DEVICE

The Allocate Device system service reserves a device for exclusive use by a process and its subprocesses. No other process can allocate the device or assign channels to it until the image that called $ALLOC exits or explicitly deallocates the device with the Deallocate Device ($DALLOC) system service.

Macro Format:

$ALLOC devnam,[phylen] ,[phybuf] ,[acmode]

High-Level Language Format:

SYS$ALLOC(devnam ,[phylen] ,[phybuf] ,[acmode])

devnam
address of a character string descriptor pointing to the device name string. The string may be either a physical device name or a logical name. If the first character in the string is an underline character (_), the name is considered a physical device name. Otherwise, a single level of logical name translation is performed and the equivalence name, if any, is used. The final name, however, cannot contain a node name unless the name is that of the host system.

phylen
address of a word to receive the length of the allocated device name string.

phybuf
address of a character string descriptor pointing to the buffer to receive the physical device name string of the allocated device. The first character in the string returned is an underline character (_).

acmode
access mode to be associated with the allocated device. The specified access mode is maximized with the access mode of the caller. Only equal or more privileged access modes can deallocate the device.

Return Status:

SYS$NORMAL
- Service successfully completed.

SYS$BUFFEROVF
- Service successfully completed. The physical name returned overflowed the buffer provided, and has been truncated.

SYS$ACCVIO
The device name string or string descriptor or physical name buffer descriptor cannot be read, or the physical name buffer cannot be written, by the caller.
SYSTEM SERVICE DESCRIPTIONS
$ALLOC - ALLOCATE DEVICE

SS$_DEVALLOC
Warning. The device is already allocated to another process.
Or, an attempt to allocate an unmounted shareable device failed
because other processes had channels assigned to the device.

SS$_DEVMOUNT
The specified device is currently mounted and cannot be
allocated; or, the device is a mailbox.

SS$_DEVNAME
No device name string was specified or the device name string
contains invalid characters.

SS$_DEVLOGNAME
The device name string has a length of 0, or has more than 63
characters.

SS$_DEVLOCAL
Warning. The device is on a remote node.

SS$_DEPVOL
An attempt was made to allocate a spooled device and the
requesting process does not have the required privilege.

SS$_DEVSUCHDEV
Warning. The specified device does not exist in the host system.

Privilege Restrictions:
A user privilege is required to allocate a spooled device.

Notes:

1. When a process calls the Assign I/O Channel ($ASSIGN) system
   service to assign a channel to a nonshareable device, such as
   a terminal or line printer, the device is implicitly
   allocated to the process.

2. This service can only be used to allocate devices that exist
   on the host system.

For an example of how to use this service, and a description of the
allocation of devices by generic device names, see Section 3.4,
"Input/Output Services."
$ASCEFC

4.4 $ASCEFC - ASSOCIATE COMMON EVENT FLAG CLUSTER

The Associate Common Event Flag Cluster system service causes a named
common event flag cluster to be associated with a process for the
execution of the current image and assigned a process-local cluster
number for use with other event flag services. If the named cluster
does not exist but the process has suitable privilege, the service
creates the cluster.

Macro Format:
$ASCEFC  efn ,name ,[prot] ,[perm]

High-Level Language Format:
SYSSASCEFC(efn ,name ,[prot] ,[perm])

efn
number of any event flag in the common cluster to be associated.
The flag number must be in the range of 64 through 95 for cluster
2 and 96 through 127 for cluster 3.

name
address of a character string descriptor pointing to the 1- to
15-character text name string for the cluster. The name is
implicitly qualified by the group number of the process issuing
the associate request.

prot
protection indicator controlling group access to the common event
flag cluster. A value of 0 (the default) indicates that any
process in the creator's group may access the cluster. A value
of 1 indicates that access is restricted to processes executing
with the creator's UIC.

perm
permanent indicator. If perm is equal to 1, the common event
cluster is marked permanent.

If perm is equal to 0, the cluster is temporary; this is the
default value.

Return Status:

SS$ NORMAL
- Service successfully completed.

SS$ ACCVIO
- The cluster name string or string descriptor cannot be read by
  the caller.

SS$ EXQUOTA
- The process has exceeded its timer queue entry quota; this quota
  controls the creation of temporary common event flag clusters.

SS$ INSFMEM
- Insufficient system dynamic memory is available to complete the
  service and the process has disabled resource wait mode with the
  Set Resource Wait Mode (SSETRWM) system service.
SYSTEM SERVICE DESCRIPTIONS
$ASCEFC - ASSOCIATE COMMON EVENT FLAG CLUSTER

SS$_ILLEFC
An illegal event flag number was specified. The cluster number must be in the range of event flags 64 through 127.

SS$_IVLOGNAM
The cluster name string has a length of 0 or has more than 15 characters.

SS$_NOPRIV
The process either does not have the privilege to create a permanent cluster; or, the protection applied to an existing cluster by its creator prohibits association.

Privilege Restrictions:

The user privilege PRMCEB is required to create a permanent common event flag cluster.

Resources Required/Returned:

Creation of temporary common event flag clusters uses the process's quota for timer queue entries (TQELM); the creation of a permanent cluster does not effect the quota. The quota is restored to the creator of the cluster when all processes associated with the cluster have disassociated.

Notes:

1. When a process associates with a common event flag cluster, that cluster's reference count is increased by 1. The reference count is decreased when a process disassociates the cluster either explicitly with the Disassociate Common Event Flag Cluster ($DACEFC) system service, or implicitly, at image exit.

   Temporary clusters are automatically deleted when their reference count goes to 0; permanent clusters must be explicitly marked for deletion with the Delete Event Flag Cluster ($DLCEFC) system service.

2. Since this service automatically creates the common event flag cluster if it does not already exist, cooperating processes need not be concerned with which process executes first to create the cluster. The first process to call $ASCEFC creates the cluster and the others associate with it regardless of the order in which they call the service.

   The initial state for all event flags in a newly-created common event flag cluster is 0.

3. If a process has already associated a cluster number with a named common event flag cluster and then issues another call to $ASCEFC with the same cluster number, the service disassociates the number from its first assignment before associating it with its second.

For an example of the $ASCEFC system service and descriptions of services that manipulate event flags, see Section 3.1, "Event Flag Services."

4-9
$ASCTIM

4.5 $ASCTIM — CONVERT BINARY TIME TO ASCII STRING

The Convert Binary Time to ASCII String system service converts an absolute or delta time from 64-bit system time format to an ASCII string. The formats of the strings returned are described in Note 2, below.

Macro Format:

$ASCTIM [timlen] ,timbuf , [timadr] , [cvtflg]

High-Level Language Format:

SYS$ASCTIM([timlen] ,timbuf , [timadr] , [cvtflg])

timlen
address of a word to receive the length of the output string returned.

timbuf
address of a character string descriptor pointing to the buffer to receive the converted string. The buffer length specified in the descriptor, together with the CVTFLG argument, controls what information is returned. See Note 3, below.

timadr
address of the 64-bit time value to be converted. If no address is specified, or is specified as 0 (the default), the current date and time are returned. A positive time value represents an absolute time. A negative time value represents a delta time. If a delta time is specified, it must be less than 10,000 days.

cvtflg
conversion indicator. A value of 1 causes only the hour, minute, second, and hundredth of second fields to be returned, depending on the length of the buffer. A value of 0 (the default) causes the full date and time to be returned, depending on the length of the buffer.

Return Status:

SS$NORMAL
Service successfully completed.

SS$IVTIME
The specified delta time is equal to or greater than 10,000 days.

Notes:

1. The $ASCTIM service executes at the access mode of the caller and does not check whether address arguments are accessible before it executes. Therefore, an access violation causes an exception condition if the input time value cannot be read or the output buffer or buffer length cannot be written.
2. The ASCII strings returned have the following formats:

Absolute Time:  dd-mmm-yyyy hh:mm:ss.cc

Delta Time:  dddd hh:mm:ss.cc

<table>
<thead>
<tr>
<th>Field</th>
<th>Length (Bytes)</th>
<th>Contents</th>
<th>Range of values</th>
</tr>
</thead>
<tbody>
<tr>
<td>dd</td>
<td>2</td>
<td>day of month</td>
<td>1 - 31</td>
</tr>
<tr>
<td>mmm</td>
<td>1</td>
<td>hyphen month</td>
<td>JAN, FEB, MAR, APR,</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td></td>
<td>MAY, JUN, JUL, AUG,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SEP, OCT, NOV, DEC</td>
</tr>
<tr>
<td>yyyy</td>
<td>4</td>
<td>year</td>
<td>1858 - 9999</td>
</tr>
<tr>
<td>blank</td>
<td>1</td>
<td>blank</td>
<td></td>
</tr>
<tr>
<td>hh</td>
<td>2</td>
<td>hour</td>
<td>00 - 23</td>
</tr>
<tr>
<td>:</td>
<td>1</td>
<td>colon</td>
<td></td>
</tr>
<tr>
<td>mm</td>
<td>2</td>
<td>minutes</td>
<td>00 - 59</td>
</tr>
<tr>
<td>:</td>
<td>1</td>
<td>colon</td>
<td></td>
</tr>
<tr>
<td>ss</td>
<td>2</td>
<td>seconds</td>
<td>00 - 59</td>
</tr>
<tr>
<td>.</td>
<td>1</td>
<td>period</td>
<td></td>
</tr>
<tr>
<td>cc</td>
<td>2</td>
<td>hundredths of</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>seconds</td>
<td></td>
</tr>
<tr>
<td>dddd</td>
<td>4</td>
<td>number of days</td>
<td>0000 - 9999</td>
</tr>
</tbody>
</table>

3. Some possible combinations of buffer length specification and CVTFLG arguments, and their results, are shown below:

<table>
<thead>
<tr>
<th>Time Value</th>
<th>Buffer Length Specified</th>
<th>CVTFLG Argument</th>
<th>Information Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>Absolute</td>
<td>24</td>
<td>0</td>
<td>date and time</td>
</tr>
<tr>
<td>Absolute</td>
<td>11</td>
<td>0</td>
<td>date</td>
</tr>
<tr>
<td>Absolute</td>
<td>11</td>
<td>1</td>
<td>time</td>
</tr>
<tr>
<td>Delta</td>
<td>17</td>
<td>0</td>
<td>days and time</td>
</tr>
<tr>
<td>Delta</td>
<td>11</td>
<td>1</td>
<td>time</td>
</tr>
</tbody>
</table>

For an example of the $ASCTIM system service, see Section 3.6, "Timer and Time Conversion Services."

**COBOL:**

```
CALL "SASCTIM" USING I_DESCRIPTOR DATE VALUE I I I.
```
$ASSIGN

4.6 $ASSIGN - ASSIGN I/O CHANNEL

The Assign I/O Channel system service (1) provides a device with an
I/O channel so that input/output operations can be performed on the
device, or (2) establishes a logical link with a remote node on a
network.

Macro Format:

$ASSIGN devnam ,chan ,[acmode] ,[mbxnam]

High-Level Language Format:

SYSSASSIGN(devnam ,chan ,[acmode] ,[mbxnam])

devnam
address of a character string descriptor pointing to the device
name string. The string may be either a physical device name or
a logical name. If the first character in the string is an
underline character (_), the name is considered a physical device
name. Otherwise, a single level of logical name translation is
performed and the equivalence name, if any, is used.

If the device name contains a double colon (::), the system
assigns a channel to the device NET0, and performs an access
function on the network.

chan
address of a word to receive the channel number assigned.

acmode
access mode to be associated with the channel. The specified
access mode is maximized with the access mode of the caller. I/O
operations on the channel can only be performed from equal and
more privileged access modes.

mbxnam
address of a character string descriptor pointing to the logical
name string for the mailbox to be associated with the device, if
any. The mailbox receives status information from the device
driver, as described in Note 2, below.

An address of 0 implies no mailbox; this is the default value.

Return Status:

SYSS$NORMAL
Service successfully completed.

SYSS$REMOTE
Service successfully completed. A logical link is established
with the target on a remote node.

SYSS$ACCVIO
The device or mailbox name string or string descriptor cannot be
read, or the channel number cannot be written, by the caller.
SYSTEM SERVICE DESCRIPTIONS
$ASSIGN - ASSIGN I/O CHANNEL

SS$_DEVALLOC
- Warning. The device is allocated to another process.

SS$_DEVNOTMBX
- A mailbox name has been specified for a device that is not a mailbox.

SS$_EXQUOTA
- The target of the assignment is on a remote node and the process has insufficient buffer quota to allocate a network control block.

SS$_INSFMEM
- The target of the assignment is on a remote node and there is insufficient system dynamic memory to complete the request.

SS$_IVDEVNAM
- No device name was specified or the device or mailbox name string contains invalid characters. If the device name is a target on a remote node, this status code indicates that the Network Connect Block has an invalid format.

SS$_IVLOGNAM
- The device or mailbox name string has a length of 0, or has more than 63 characters.

SS$_NOIOCHAN
- No I/O channel is available for assignment.

SS$_NOLINKS
- No logical network links are available.

SS$_NOPRIV
- The process does not have the privilege to perform network operations.

SS$_NOSUCHDEV
- Warning. The specified device or mailbox does not exist.

SS$_NOSUCHNODE
- The specified network node is nonexistent or unavailable.

SS$_REJECT
- The network connect was rejected by NSP or by the partner at the remote node; or, the target image exited before the connect confirm could be issued.

Resources Required/Returned:

System dynamic memory is required if the target device is on a remote system.
Notes:

1. For details on how to use $ASSIGN in conjunction with network operations, see the DECnet-VAX User's Guide.

2. Only the owner of a device can associate a mailbox with the device (the owner is the process that has allocated the device, either implicitly or explicitly). Then, the device driver can send messages containing status information to the mailbox, as in the following cases:
   - If the device is a terminal, a message indicates dialup, hangup, or the reception of unsolicited input.
   - If the target is on a network, the message may indicate the network connect or initiate, or whether the line is down.
   - If the device is a line printer, the message indicates that the printer is offline.

   For details on the message format and the information returned, see the VAX/VMS I/O User's Guide.

3. Channels remain assigned until they are explicitly reassigned with the Deassign I/O Channel ($DASSGN) system service, or until the image that assigned the channel exits.

4. The $ASSIGN service establishes a path to a device, but does not check whether the caller can actually perform input/output operations to the device. Privilege and protection restrictions may be applied by the device drivers. For details on how the system controls access to devices, see the VAX/VMS I/O User's Guide.

For examples of how to use $ASSIGN to assign channels for input/output operations, see Section 3.4, "Input/Output Services."
4.7 $BINTIM - CONVERT ASCII STRING TO BINARY TIME

The Convert ASCII String to Binary Time system service converts an ASCII string to an absolute or delta time value in the system 64-bit time format suitable for input to the Set Timer ($SETIMR) or Schedule Wakeup ($SCHDWK) system services.

Macro Format:

$BINTIM timbuf ,timadr

High-Level Language Format:

SYS$BINTIM(timbuf ,timadr)

timbuf

address of a character string descriptor pointing to the buffer containing the absolute or delta time to be converted. The required formats of the ASCII strings are described in the Notes, below.

If a delta time is specified, it must be less than 10,000 days.

timadr

address of a quadword that is to receive the converted time in 64-bit format.

Return Status:

SS$ NORMAL

Service successfully completed.

SS$ IVTIME

The syntax of the specified ASCII string is invalid, or the time component is out of range.

Notes:

1. The $BINTIM service executes at the access mode of the caller and does not check whether address arguments are accessible before it executes. Therefore, an access violation causes an exception condition if the input buffer or buffer descriptor cannot be read or the output buffer cannot be written.
2. The required ASCII input strings have the format:

Absolute Time: dd-mmm-yyyy hh:mm:ss.cc
Delta Time: dddd hh:mm:ss.cc

<table>
<thead>
<tr>
<th>Field</th>
<th>Length (Bytes)</th>
<th>Contents</th>
<th>Range of values</th>
</tr>
</thead>
<tbody>
<tr>
<td>dd</td>
<td>2</td>
<td>day of month</td>
<td>1 - 31</td>
</tr>
<tr>
<td>-</td>
<td>1</td>
<td>hyphen</td>
<td>Required syntax</td>
</tr>
<tr>
<td>mmm</td>
<td>3</td>
<td>month</td>
<td>JAN, FEB, MAR, APR,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MAY, JUN, JUL, AUG,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SEP, OCT, NOV, DEC</td>
</tr>
<tr>
<td>-</td>
<td>1</td>
<td>hyphen</td>
<td>Required syntax</td>
</tr>
<tr>
<td>yyyy</td>
<td>4</td>
<td>year</td>
<td>1858 - 9999</td>
</tr>
<tr>
<td>blank</td>
<td>n</td>
<td>blank</td>
<td>Required syntax (one or more blanks)</td>
</tr>
<tr>
<td>hh</td>
<td>2</td>
<td>hour</td>
<td>00 - 23</td>
</tr>
<tr>
<td>:</td>
<td>1</td>
<td>colon</td>
<td>Required syntax</td>
</tr>
<tr>
<td>mm</td>
<td>2</td>
<td>minutes</td>
<td>00 - 59</td>
</tr>
<tr>
<td>:</td>
<td>1</td>
<td>colon</td>
<td>Required syntax</td>
</tr>
<tr>
<td>ss</td>
<td>2</td>
<td>seconds</td>
<td>00 - 59</td>
</tr>
<tr>
<td>.</td>
<td>1</td>
<td>period</td>
<td>Required syntax</td>
</tr>
<tr>
<td>cc</td>
<td>2</td>
<td>hundredths</td>
<td>of seconds</td>
</tr>
<tr>
<td>dddd</td>
<td>4</td>
<td>number of days (in 24-hour units)</td>
<td>0000 - 9999</td>
</tr>
</tbody>
</table>

3. The following syntax rules apply to the coding of the ASCII input string:

- Any of the fields of the date and time can be omitted.

For absolute time values, the $BINTIM$ service supplies the current system date and time for nonspecified fields. Trailing fields can be truncated. If leading fields are omitted, the punctuation (hyphens, blanks, colons, periods) must be specified. For example, the string

```
-- 12:00:00.00
```

results in an absolute time of 12:00 on the current day.

For delta time values, the $BINTIM$ service defaults nonspecified fields to 0. Trailing fields can be truncated. If leading fields are omitted from the time value, the punctuation (blanks, colons, periods) must be specified. For example, the string

```
0 ::10
```

results in a delta time of 10 seconds.

- For both absolute and delta time values, there can be any number of leading blanks, and any number of blanks between fields normally delimited by blanks. However, there can be no embedded blanks within either the date or time fields.
4.8 $BRDCST - BROADCAST

The Broadcast system service writes a message to one or more terminals.

Macro Format:

$BRDCST msgbuf, [devnam]

High-Level Language Format:

SYS$BRDCST(msgbuf, [devnam])

msgbuf
address of a character string descriptor pointing to the text of the message to be broadcast. The maximum length of the message is 250 bytes.

devnam
address of a character string descriptor pointing to the name of the terminal that is to receive the message. The string may be either a physical device name or a logical name. If the first character in the string is an underscore character (\_), the name is considered a physical device name. Otherwise, a single level of logical name translation is performed and the equivalence name, if any, is used.

If this argument is omitted, or specified as 0, then the message is broadcast to all terminals.

If the first longword in the descriptor contains a 0, the message is sent to all terminals that are currently allocated to processes.

Return Status:

SS$ _NORMAL
Service successfully completed.

SS$ _ACCVIO
The message buffer or buffer descriptor, or the device name string or string descriptor, cannot be read by the caller.

SS$ _DEVOFFLINE
The specified terminal is offline, has disabled broadcast message reception, has enabled passall mode, or is not a terminal.

SS$ _EXQUOTA
The process has exceeded its buffer space quota and has disabled resource wait mode with the Set Resource Wait Mode ($SETRWM) system service.

SS$ _INSFMEM
Insufficient system dynamic memory is available to complete the request and the process has disabled resource wait mode with the Set Resource Wait Mode ($SETRWM) system service.
SSS_NOPRIV
- The process does not have the privilege to broadcast messages.

SSS_NOSUCHDEV
- Warning. The specified terminal does not exist, or it cannot receive the message.

Privilege Restrictions:

The user privilege OPER is required to broadcast a message to more than one terminal, or to broadcast a message to a terminal that is allocated to any other user.

Resources Required/Returned:

This service requires system dynamic memory, and uses the process's buffered I/O byte count quota (BYTLM) to buffer the message while the service executes.

Notes:

1. The service does not return control to the caller until all specified terminals have received the broadcast message.

2. The message is displayed at all specified terminals immediately, regardless of the current state of the terminal (reading or writing). Each terminal is then returned to the state it was in prior to the reception of the message. The message is preceded and followed by a carriage return/line feed.

However, a terminal cannot receive a broadcast message if it is not in use as an interactive terminal.
4.9 $CANCEL - CANCEL I/O ON CHANNEL

The Cancel I/O On Channel system service cancels all pending I/O requests on a specific channel. In general, this includes all I/O requests that are queued as well as the request currently in progress.

Macro Format:

$CANCEL chan

High-Level Language Format:

SYS$CANCEL(chan)

chan
number of the I/O channel on which I/O is to be canceled.

Return Status:

SS$ NORMAL
Service successfully completed.

SS$ EXQUOTA
The process has exceeded its quota for direct I/O and has disabled resource wait mode with the Set Resource Wait Mode ($SETRWM) system service.

SS$ INSFMEM
Insufficient system dynamic memory is available to cancel the I/O, and the process has disabled resource wait mode with the Set Resource Wait Mode ($SETRWM) system service.

SS$ IVCHAN
An invalid channel was specified, that is, a channel number of 0 or a number larger than the number of channels available.

SS$ NOPRIV
The specified channel is not assigned, or was assigned from a more privileged access mode.

Privilege Restrictions:

I/O can be canceled only from an access mode equal to or more privileged than the access mode from which the original channel assignment was made.

Resources Required/Returned:

The Cancel I/O On Channel system service requires system dynamic memory and uses the process's direct I/O limit (DIOLM) quota.
SYSTEM SERVICE DESCRIPTIONS
$CANCEL - CANCEL I/O ON CHANNEL

Notes:

1. When a request currently in progress is canceled, the driver is notified immediately. Actual cancellation may or may not occur immediately depending on the logical state of the driver. When cancellation does occur, the same action as that taken for queued requests is performed:
   a. The specified event flag is set.
   b. The first word of the I/O status block, if specified, is set to SS$CANCEL.
   c. The AST, if specified, is queued.

   Proper synchronization between this service and the actual canceling of I/O requests requires the issuing process to wait for I/O completion in the normal manner and then note that the I/O has been canceled.

2. If the I/O operation is a virtual I/O operation involving a disk or tape ACP, the I/O cannot be canceled. In the case of a magnetic tape, however, cancellation may occur if the device driver is hung.

3. Outstanding I/O requests are automatically canceled at image exit.

For an example of the $CANCEL system service and additional information on system services that perform device-dependent I/O operations, see Section 3.4, "Input/Output Services."
4.10 $CANEXH - CANCEL EXIT HANDLER

The Cancel Exit Handler system service deletes an exit control block from the list of control blocks for the calling access mode. Exit control blocks are declared by the Declare Exit Handler ($DCLEXH) system service, and are queued according to access mode in a last-in first-out order.

Macro Format:

$CANEXH [desblk]

High-Level Language Format:

SYS$CANEXH([desblk])

desblk
address of the control block describing the exit handler to be canceled. If not specified, or specified as 0, all exit control blocks are canceled for the current access mode.

Return Status:

SS$NORMAL
- Service successfully completed.

SS$ACCVIO
- The first longword of the exit control block or the first longword of a previous exit control block in the list cannot be read by the caller, or the first longword of the preceding control block cannot be written by the caller.

SS$NOHANDLER
- Warning. The exit handler specified does not exist.
$SCANTIM

4.11 $SCANTIM - CANCEL TIMER REQUEST

The Cancel Timer Request system service cancels all or a selected subset of the Set Timer requests previously issued by the current image executing in a process. Cancellation is based on the request identification specified in the Set Timer ($SETIMR) system service. If more than one timer request was given the same request identification, they are all canceled.

Macro Format:

$SCANTIM [reqidt], [acmode]

High-Level Language Format:

SYS$SCANTIM([reqidt], [acmode])

reqidt
request identification of the timer request(s) to be canceled. A value of 0 (the default) indicates that all timer requests are to be canceled.

acmode
access mode of the request(s) to be canceled. The access mode is maximized with the access mode of the caller. Only those timer requests issued from an access mode equal to or less privileged than the resultant access mode are canceled.

Return Status:

$SS$NORMAL
Service successfully completed.

Privilege Restrictions:

Timer requests can be canceled only from access modes equal to or more privileged than the access mode from which the requests were issued.

Resources Required/Returned:

Canceled timer requests are restored to the process's quota for timer queue entries (TQELM quota).

Note:

Outstanding timer requests are automatically canceled at image exit.

For an example of the $SCANTIM system service, and additional information on timer scheduled requests, see Section 3.6, "Timer and Time Conversion Services."
4.12 $CANWAK - CANCEL WAKEUP

The Cancel Wakeup system service removes all scheduled wakeu up requests for a process from the timer queue, including those made by the caller or by other processes. Scheduled wakeu up requests are made with the Schedule Wakeup ($SCHDWK) system service.

Macro Format:

$CANWAK [pidadr], [prcnam]

High-Level Language Format:

SYS$CANWAK([pidadr], [prcnam])

pidadr
address of a longword containing the process identification of the process for which wakeups are to be canceled.

prcnam
address of a character string descriptor pointing to the process name string. The process name is implicitly qualified by the group number of the process issuing the cancel wakeup request.

If neither a process identification nor a process name is specified, scheduled wakeu up requests for the caller are canceled. For details on how the service interprets the PIDADR and PRCNAM arguments, see Table 3-3. Table 3-3 is in Section 3.5, "Process Control Services."

Return Status:

SS$ NORMAL
Service successfully completed.

SS$ ACCVIO
The process name string or string descriptor cannot be read, or the process identification cannot be written, by the caller.

SS$ IVLOGNAM
The process name string has a length of 0, or has more than 15 characters.

SS$ NONEXPR
Warning. The specified process does not exist, or an invalid process identification was specified.

SS$ NOPRIV
The process does not have the privilege to cancel wakeups for the specified process.
Privilege Restrictions:

User privileges are required to cancel scheduled wakeup requests for:

- Other processes in the same group (GROUP privilege)
- Any other process in the system (WORLD privilege)

Resources Required/Returned:

Canceled wakeup requests are restored to the process's AST limit quota (ASTLM).

Notes:

1. Pending wakeup requests issued by the current image are automatically canceled at image exit.

2. This service only cancels wakeup requests that have been scheduled; it does not cancel wakeup requests made with the Wake Process ($WAKE) system service.

For an example of the SCANWAK system service, see Section 3.6, "Timer and Time Conversion Services." For more information on process hibernation and waking, see Section 3.5, "Process Control Services."
4.13 $CLREF – CLEAR EVENT FLAG

The Clear Event Flag system service sets an event flag in a local or common event flag cluster to 0.

Macro Format:

$CLREF efn

High-Level Language Format:

SYSS$CLREF(efn)

efn
   number of the event flag to be cleared.

Return Status:

SS$ _WASCLR
   Service successfully completed. The specified event flag was previously 0.

SS$ _WASSET
   Service successfully completed. The specified event flag was previously 1.

SS$ _ILLEFC
   An illegal event flag number was specified.

SS$ _UNASEFC
   The process is not associated with the cluster containing the specified event flag.

Note:

For an example of the $CLREF system service, see Section 3.1, "Event Flag Services."
$CMEXEC

4.14 $CMEXEC - CHANGE TO EXECUTIVE MODE

The Change to Executive Mode system service allows a process to change its access mode to executive, execute a specified routine, and then return to the access mode in effect before the call was issued.

Macro Format:

$CMEXEC routin,[arglst]

High-Level Language Format:

SYSS$CMEXEC(routin,[arglst])

routin
address of the routine to be executed in executive mode.

arglst
address of the argument list to be supplied to the routine, if any.

Return Status:

SS$_NOPRIV
The process does not have the privilege to change mode to executive.

All other values returned are from the routine executed.

Privilege Restrictions:

A process can call this service if:

- It has the user privilege CMEXEC.
- It is currently executing in either executive or kernel mode.

Note:

The $CMEXEC system service uses standard procedure calling conventions to pass control to the specified routine. If no argument list is specified, the argument pointer (AP) contains a 0, unless it is modified by the caller. The routine must exit with a RET instruction.
4.15 $CMKRNL - CHANGE TO KERNEL MODE

The Change to Kernel Mode system service allows a process to change its access mode to kernel, execute a specified routine, and then return to the access mode in effect before the call was issued.

Macro Format:

$CMKRNL routin ,[arglst]

High-Level Language Format:

SYSS$CMKRNL(routin ,[arglst])

routin
address of the routine to be executed in kernel mode.

arglst
address of the argument list to be supplied to the routine, if any.

Return Status:

SS$_NOPRIV
The process does not have the privilege to change mode to kernel.

All other values returned are from the routine executed.

Privilege Restrictions:

A process can call this service if:

- It has the user privilege CMKRNL.
- It is currently executing in either executive or kernel mode.

Note:

The $CMKRNL system service uses standard procedure calling conventions to pass control to the specified routine. If no argument list is specified, the argument pointer (AP) contains a 0, unless it is modified by the caller. The routine must exit with a RET instruction.
$CNTREG

4.16 $CNTREG - CONTRACT PROGRAM/CONTROL REGION

The Contract Program/Control Region system service deletes a specified number of pages from the current end of the program or control region of a process's virtual address space. The deleted pages become inaccessible; any references to them cause access violations.

Macro Format:

$CNTREG pagcnt,[retadr],[acmode],[region]

High-Level Language Format:

SYS$CNTREG(pagcnt,[retadr],[acmode],[region])

pagcnt
number of pages to be deleted from the current end of the program or control region.

retadr
address of a 2-longword array to receive the virtual addresses of the starting page and ending page of the deleted area.

acmode
access mode of the owner of the pages to be deleted. The specified access mode is maximized with the access mode of the caller.

region
region indicator. A value of 0 (the default) indicates that the program region (P0 region) is to be contracted, and a value of 1 indicates that the control region (P1 region) is to be contracted.

Return Status:

SS$ NORMAL
Service successfully completed.

SS$ ACCVIO
The return address array cannot be written by the caller.

SS$ ILLPAGCNT
The specified page count was less than 1.

SS$ PAGOWNVIO
A page in the specified range is owned by a more privileged access mode.
Notes:

1. If an error occurs while deleting pages, the return array, if requested, indicates the pages that were successfully deleted before the error occurred. If no pages were deleted, both longwords in the return address array contain a -1.

2. The $CNTREG system service can delete pages only from the current end of the process's program or control region. To delete a specific range of pages in either region, use the Delete Virtual Address Space ($DELTVA) system service.

For an example of the $CNTREG system service and additional details on page creation and deletion, see Section 3.8.2, "Increasing and Decreasing Virtual Address Space."
$CRELOG

4.17 $CRELOG - CREATE LOGICAL NAME

The Create Logical Name system service inserts a logical name and its equivalence name into the process, group, or system logical name table. If the logical name already exists in the respective table, the new definition supersedes the old.

Macro Format:

$CRELOG [tblflg],lognam,eqlnam,[acmode]

High-Level Language Format:

SYS$CRELOG([tblflg],lognam,eqlnam,[acmode])

tblflg
logical name table number. A value of 0 indicates the system table (this is the default value), 1 indicates the group table, and 2 indicates the process logical name table.

lognam
address of a character string descriptor pointing to the logical name string. (upper case)

eqlnam
address of a character string descriptor pointing to the equivalence name string.

acmode
access mode to be associated with the logical name table entry. Access modes only qualify names in the process logical name table. The specified access mode is maximized with the access mode of the caller.

Return Status:

SS$ _NORMAL
Service successfully completed. A new name was entered in the specified logical name table.

SS$ _SUPERSEDE
Service successfully completed. A new equivalence name replaced a previous equivalence name in the specified logical name table.

SS$ _ACCVO
The logical name or equivalence name string or string descriptor cannot be read by the caller.

SS$ _INSF MEM
Insufficient system dynamic memory is available to allocate a group or system logical name table entry or the process has exceeded its limit for process logical name table entries. The code is only returned if the process has disabled resource wait mode with the Set Resource Wait Mode ($SETRWM) system service.
$CRELOG - CREATE LOGICAL NAME

**SS$ IVLOGNAM**
- The logical name or equivalence name string has a length of 0, or has more than 63 characters.

**SS$ IVLOGTAB**
- An invalid logical name table number was specified.

**SS$ NOPRIV**
- The process does not have the privilege to place an entry in the specified logical name table.

**Privilege Restrictions:**

The user privileges GRPNAM and SYSNAM are required to place entries in the system and group logical name tables, respectively.

**Resources Required/Returned:**

1. Up to 5 pages of memory are available in the control region of a process's virtual address space to store names in the process logical name table.

2. Creation of logical names for the group and system logical name tables requires system dynamic memory.

**Note:**

Logical names can also be created from the command stream, with the ASSIGN, DEFINE, ALLOCATE, and MOUNT commands.

For examples of the $CRELOG system service, and details on logical name translation and deletion, see Section 3.3, "Logical Name Services."
$CREMBX

4.18 $CREMBX - CREATE MAILBOX AND ASSIGN CHANNEL

The Create Mailbox and Assign Channel system service creates a virtual mailbox device named MBN: and assigns an I/O channel to it. The system provides the unit number, n, when it creates the mailbox.

Macro Format:

$CREMBX [prmflg] ,chan ,[maxmsg] ,[bufquo] ,[promsk] 
,[acmode] ,[lognam]

High-Level Language Format:

SYSS$CREMBX([prmflg] ,chan ,[maxmsg] ,[bufquo] ,[promsk] 
,[acmode] ,[lognam])

prmflg
permanent indicator. A value of 1 indicates that a permanent mailbox is to be created. The logical name, if specified, is entered in the system logical name table. A value of 0 (the default) indicates a temporary mailbox. The logical name, if specified, is entered in the group logical name table.

chan
address of a word to receive the channel number assigned.

maxmsg
number indicating the maximum size of messages that can be sent to the mailbox. If not specified, or specified as 0, the system provides a default value.

bufquo
number of bytes of system dynamic memory that can be used to buffer messages sent to the mailbox. For a temporary mailbox, this value must be less than or equal to the process buffer quota. If not specified, or specified as 0, the system provides a default value.

promsk
numeric value representing the protection mask for the mailbox.

The mask contains four 4-bit fields:

<table>
<thead>
<tr>
<th>15</th>
<th>11</th>
<th>7</th>
<th>3</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORLD</td>
<td>GROUP</td>
<td>OWNER</td>
<td>SYSTEM</td>
<td></td>
</tr>
</tbody>
</table>

Bits read from right to left in each field, when clear, indicate that read, write, extend and delete privileges, in that order, are granted to the particular category of user.

Only read and write privileges are meaningful for mailbox protection.

If not specified, or specified as 0, read and write privileges are granted to all users.
acmode
access mode to be associated with the channel to which the mailbox is assigned. The access mode is maximized with the access mode of the caller.

lognam
address of a character string descriptor pointing to the logical name string for the mailbox. The logical name is entered into the group logical name table (if it is a temporary mailbox) or the system logical name table (if it is a permanent mailbox). In either case, the MBn: name is entered as the equivalence name (the first character in the equivalence name string is an underline character [_]). Processes can use the logical name to assign I/O channels to the mailbox.

Return Status:

SS$_NORMAL
- Service successfully completed.

SS$_SUPERSEDE
- Service successfully completed. A new equivalence name replaced a previous equivalence name for the mailbox logical name.

SS$_ACCVIO
- The logical name string or string descriptor cannot be read, or the channel number cannot be written, by the caller.

SS$_EXQUOTA
- The process has exceeded its buffered I/O byte count quota.

SS$_INSPMEM
- Insufficient system dynamic memory is available to complete the service.

SS$_IVLOGNAM
- The logical name string has a length of 0 or has more than 63 characters.

SS$_NOICHAN
- No I/O channel is available for assignment.

SS$_NOPRIV
- The process does not have the privilege to create either a temporary or a permanent mailbox.

Privilege Restrictions:

The user privileges TMPMBX and PRMMBX are required to create temporary and permanent mailboxes, respectively.
SYSTEM SERVICE DESCRIPTIONS
$CREMBX - CREATE MAILBOX AND ASSIGN CHANNEL

Resources Required/Returned:

1. System dynamic memory is required for the allocation of a
device data base for the mailbox and for an entry in the
logical name table, if a logical name is specified.

2. When a temporary mailbox is created, the process's buffered
I/O byte count (BYTLM) quota is reduced by the amount
specified in the BUFQUO argument. The size of the mailbox
unit control block, and the logical name (if one is
specified) are also subtracted from the quota. The quota is
returned to the process when the mailbox is deleted.

Notes:

1. After a mailbox is created, the creating process and other
processes can assign additional channels to it by calling the
Assign I/O Channel ($ASSIGN) system service. The system
maintains a reference count of the number of channels
assigned to a mailbox; the count is decreased whenever a
channel is reassigned with the Deassign I/O Channel ($DASSGN)
system service or when the image that assigned the channel
exits. If it is a temporary mailbox, it is deleted when
there are no more channels assigned. Permanent mailboxes
must be explicitly marked for deletion with the Delete
Mailbox ($DELMBX) system service.

2. A mailbox is treated as a shareable device; it cannot,
however, be mounted or allocated.

3. Mailboxes are assigned sequentially increasing unit numbers
(from 1 to a maximum of 65,535) as they are created. When
all unit numbers have been used, the system starts numbering
again at unit 1.

4. A process can obtain the unit number of the created mailbox
by calling the Get I/O Channel Information ($GETCHN) system
service.

5. Default values for the maximum message size and the buffer
quota (an appropriate multiple of the message size) are
determined for a specific system during system generation.

For an example of mailbox creation and input/output operations to it,
see Section 3.4.13, "Mailboxes."
4.19 $CREPRC - CREATE PROCESS

The Create Process system service allows a process to create another process. The created process can be either a subprocess or a detached process.

A detached process is a fully-independent process. For example, the process that the system creates when a user logs in is a detached process. A subprocess, on the other hand, is related to its creator in a tree like structure; it receives a portion of the creating process's resource quotas and must terminate before the creating process. The specification of the UIC argument controls whether the created process is a subprocess or a detached process.

Macro Format:

$CREPRC [pidadr], [image], [input], [output], [error], [prvadr], [quota], [prcnam], [baspri], [uic], [mbxunt], [stsflg]

High-Level Language Format:

SYS$CREPRC([pidadr], [image], [input], [output], [error], [prvadr], [quota], [prcnam], [baspri], [uic], [mbxunt], [stsflg])

pidadr
address of a longword to receive the process identification number assigned to the created process.

image
address of a character string descriptor pointing to the file specification of the image to be activated in the created process. The image name can have a maximum of 63 characters.

input
address of a character string descriptor pointing to the equivalence name string to be associated with the logical name SYS$INPUT in the logical name table for the created process. The equivalence name string can have a maximum of 63 characters.

output
address of a character string descriptor pointing to the equivalence name string to be associated with the logical name SYS$OUTPUT in the logical name table for the created process. The equivalence name string can have a maximum of 63 characters.

error
address of a character string descriptor pointing to the equivalence name string to be associated with the logical name SYS$ERROR in the logical name table for the created process. The equivalence name string can have a maximum of 63 characters.

prvadr
address of a 64-bit mask defining privileges for the created process. The mask is formed by ORing bit settings corresponding to specific privileges. The $PRVDEF macro defines the following symbolic names for the bit settings:
<table>
<thead>
<tr>
<th>Name</th>
<th>Privilege</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRVSV_ALLSPPOOL</td>
<td>Allocate a spooled device</td>
</tr>
<tr>
<td>PRVSV_BUGCHK</td>
<td>Make bug check error log entries</td>
</tr>
<tr>
<td>PRVSV_CMEXEC</td>
<td>Change mode to executive</td>
</tr>
<tr>
<td>PRVSV_CMRNL</td>
<td>Change mode to kernel</td>
</tr>
<tr>
<td>PRVSV_DETACH</td>
<td>Create detached processes</td>
</tr>
<tr>
<td>PRVSV_DIAGNOSE</td>
<td>Diagnose devices</td>
</tr>
<tr>
<td>PRVSV_EXQUOTA</td>
<td>Exceed quotas</td>
</tr>
<tr>
<td>PRVSV_GROUP</td>
<td>Group process control</td>
</tr>
<tr>
<td>PRVSV_GRPNAME</td>
<td>Place name in group logical name table</td>
</tr>
<tr>
<td>PRVSV_LOG_IO</td>
<td>Perform logical I/O operations</td>
</tr>
<tr>
<td>PRVSV_MOUNT</td>
<td>Issue mount volume QIO</td>
</tr>
<tr>
<td>PRVSV_NETMBX</td>
<td>Create a network device</td>
</tr>
<tr>
<td>PRVSV_NOACT</td>
<td>Create processes for which no accounting is done</td>
</tr>
<tr>
<td>PRVSV_OPER</td>
<td>All operator privileges</td>
</tr>
<tr>
<td>PRVSV_PHY_IO</td>
<td>Perform physical I/O operations</td>
</tr>
<tr>
<td>PRVSV_PRMCEB</td>
<td>Create permanent common event flag clusters</td>
</tr>
<tr>
<td>PRVSV_PRMGBL</td>
<td>Create permanent global sections</td>
</tr>
<tr>
<td>PRVSV_PRMMBX</td>
<td>Create permanent mailboxes</td>
</tr>
<tr>
<td>PRVSV_PSAPM</td>
<td>Change process swap mode</td>
</tr>
<tr>
<td>PRVSV_SETPRI</td>
<td>Set any process priority</td>
</tr>
<tr>
<td>PRVSV_SETPRV</td>
<td>Set any process privileges</td>
</tr>
<tr>
<td>PRVSV_SYSGBL</td>
<td>Create system global sections</td>
</tr>
<tr>
<td>PRVSV_SYSNAM</td>
<td>Place name in system logical name table</td>
</tr>
<tr>
<td>PRVSV_TMPMBX</td>
<td>Create temporary mailboxes</td>
</tr>
<tr>
<td>PRVSV_VOLPRO</td>
<td>Override volume protection</td>
</tr>
<tr>
<td>PRVSV_WORLD</td>
<td>World process control</td>
</tr>
</tbody>
</table>

The user privilege SETPRV is required to grant a process any privileges higher than one's own. If the caller does not have this privilege, the mask is minimized with the current privileges of the creating process, that is, any privileges the creator does not have are not granted but no error status code is returned.

**quota**

address of a list of values assigning resource quotas to the created process. If no address is specified, or the address is specified as 0, the system supplies default values for the resource quotas.

The format of the quota list and considerations for specifying quota values are described in Section 4.19.1. The specific quotas, their defaults, and their minimum values, are listed in Section 4.19.2.

**prcnam**

address of a character string descriptor pointing to a 1- to 15-character process name string to be assigned to the created process. The process name is implicitly qualified by the group number of the caller, if a subprocess is created, or by the group number in the UIC argument, if a detached process is created.

**baspri**

numeric value indicating the base priority to be assigned to the created process. The priority must be in the range of 0 to 31, where 31 is the highest priority level and 0 is the lowest. Normal priorities are in the range 0 through 15, and time-critical priorities are in the range 16 through 31.
SYSTEM SERVICE DESCRIPTIONS
$CREPRC - CREATE PROCESS

If not specified, the base priority for the created process is 2.

The user privilege ALTPRI is required to set a priority higher than one's own. If the caller does not have this privilege, the specified base priority is compared with the caller's priority and the lower of the two values is used.

**uic**

numeric value representing the user identification code (UIC) of the created process. This argument also indicates whether a process is a subprocess or a detached process.

If not specified, or specified as 0 (the default), it indicates that the created process is a subprocess; the subprocess has the same UIC as the creator.

If a nonzero value is specified, it indicates that the created process is a detached process. The specified value is interpreted as a 32-bit octal number, with two 16-bit fields:

- bits 0-15 member number
- bits 16-31 group number

The user privilege DETACH is required to create a detached process.

**mbxunr**

unit number of a mailbox to receive a termination message when the created process is deleted. If not specified, or specified as 0 (the default), the system sends no termination message when it deletes the process. The format of the message is described in Note 2, below.

**cstflag**

32-bit status flag indicating options selected for the created process. The flag bits, when set, have the following meanings:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Disable resource wait mode</td>
</tr>
<tr>
<td>1</td>
<td>Enable system service failure exception mode</td>
</tr>
<tr>
<td>2</td>
<td>Inhibit process swapping (PSWAPM privilege required)</td>
</tr>
<tr>
<td>3</td>
<td>Do not perform accounting (NOACNT privilege required)</td>
</tr>
<tr>
<td>4</td>
<td>Batch (non-interactive) process</td>
</tr>
<tr>
<td>5</td>
<td>Force process to hibernate before it executes the image</td>
</tr>
<tr>
<td>6</td>
<td>Provide detached process executing LOGIN image with authorization file attributes of the creator; do not check authorization file</td>
</tr>
<tr>
<td>7</td>
<td>Process is a network connect object (NETMBX privilege required)</td>
</tr>
<tr>
<td>8-31</td>
<td>Reserved. These bits must be 0.</td>
</tr>
</tbody>
</table>

**Return Status:**

- **SS$ NORMAL**
  
  Service successfully completed.

- **SS$ ACCVIO**

  The caller cannot read a specified input string or string descriptor, the privilege list, or the quota list. Or, the caller cannot write the process identification.

4-37
SS$_DUPLNAM
The process name specified duplicates one already specified within that group.

SS$_EXQUOTA
1. The process has exceeded its quota for the creation of subprocesses.
2. A quota value specified for the creation of a subprocess exceeds the creator's corresponding quota; or, the quota is deductible and the remaining quota for the creator would be less than the minimum.

SS$_INSPMEM
Insufficient system dynamic memory is available to complete the service and the process has disabled resource wait mode with the Set Resource Wait Mode ($SETRWM) system service.

SS$_IVLOGNAM
The specified process name has a length of 0 or has more than 15 characters.

SS$_IVQUOOTAL
The quota list is not in the proper format.

SS$_IVSTSFLG
A reserved status flag was set.

SS$_NOPRIV
The caller has violated one of the privilege restrictions listed below.

Privilege Restrictions:

User privileges are required to:
1. Create detached processes (DETACH privilege)
2. Set a created subprocess's base priority higher than one's own (ALTPRI privilege)
3. Grant a process user privileges that the caller does not have (SETPRV privilege)
4. Disable either process swap mode (PSWAPM privilege) or accounting functions (NOACNT privilege) for the created process
5. Create a network connect object (NETMBX privilege)

Resources Required/Returned:

1. The number of subprocesses that a process can create is controlled by the subprocess quota (PRCLM); the quota amount is returned when a subprocess is deleted.
2. The Create Process system service requires system dynamic memory.
SYSTEM SERVICE DESCRIPTIONS
$CREPRC - CREATE PROCESS

3. When a subprocess is created, certain of the quotas granted to it either specifically or by default are deducted from the quotas of the creator, and may be returned to the creator when the subprocess is deleted. Sections 4.19.1 through 4.19.3 describe how quotas are determined in process creation.

Notes:

1. Some error conditions are not detected until the created process executes. These conditions include an invalid or nonexistent image; invalid SYS$INPUT, SYS$OUTPUT, or SYS$ERROR logical name equivalences; and inadequate quotas or insufficient privilege to execute the requested image.

2. If a mailbox unit is specified, the mailbox is not used until the created process actually terminates. At that time, a $ASSIGN system service is issued for the mailbox in the context of the terminating process and an accounting message is sent to the mailbox. If the mailbox no longer exists, cannot be assigned, or is full, the error is treated as if no mailbox had been specified.

The message is sent before the process rundown is initiated but after the process name has been set to null. Thus, a significant interval of time can occur between the sending of the termination message and the final deletion of the process.

To receive the message, the caller must issue a read to the mailbox. When the I/O completes, the second longword of the I/O status block, if one is specified, contains the process identification of the deleted process.

Symbolic names for offsets of fields within the accounting message are defined in the $ACCDEF macro. The offsets, their symbolic names, lengths, and the contents of each field are listed below.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Name</th>
<th>Length</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>ACC$W_MSGTYP</td>
<td>word</td>
<td>MSG$_DELPROC</td>
</tr>
<tr>
<td>2</td>
<td>ACC$L_FINALSTS</td>
<td>word</td>
<td>not used</td>
</tr>
<tr>
<td>4</td>
<td>ACC$L_PID</td>
<td>longword</td>
<td>Exit status code</td>
</tr>
<tr>
<td>8</td>
<td>ACC$L_TERMIN</td>
<td>longword</td>
<td>Process identification</td>
</tr>
<tr>
<td>12</td>
<td>ACC$QTERMTIME</td>
<td>longword</td>
<td>Not used</td>
</tr>
<tr>
<td>16</td>
<td>ACC$ACCOUNT</td>
<td>quadword</td>
<td>Current time in system format at process termination</td>
</tr>
<tr>
<td>24</td>
<td>ACC$T_ACCOUNT</td>
<td>8 bytes</td>
<td>Account name for process, blank filled</td>
</tr>
<tr>
<td>32</td>
<td>ACC$T_USERNAME</td>
<td>12 bytes</td>
<td>User name, blank filled</td>
</tr>
<tr>
<td>44</td>
<td>ACC$L_CPU</td>
<td>longword</td>
<td>CPU time used by the process, in 10-millisecond units</td>
</tr>
<tr>
<td>48</td>
<td>ACC$L_PAGEFLTS</td>
<td>longword</td>
<td>Number of page faults incurred by the process in its lifetime</td>
</tr>
<tr>
<td>52</td>
<td>ACC$L_PGFLP</td>
<td>longword</td>
<td>Peak paging file usage</td>
</tr>
<tr>
<td>56</td>
<td>ACC$L_WSPEAK</td>
<td>longword</td>
<td>Peak working set size</td>
</tr>
<tr>
<td>60</td>
<td>ACC$L_BIOCNT</td>
<td>longword</td>
<td>Count of buffered I/O operations performed by the process</td>
</tr>
</tbody>
</table>

4-39
SYSTEM SERVICE DESCRIPTIONS
$CREPRC - CREATE PROCESS

<table>
<thead>
<tr>
<th>Offset</th>
<th>Name</th>
<th>Length</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>ACC$L_DIOCNT</td>
<td>longword</td>
<td>Count of direct I/O operations performed by the process</td>
</tr>
<tr>
<td>68</td>
<td>ACC$L_VOLUMES</td>
<td>longword</td>
<td>Count of volumes mounted by the process</td>
</tr>
<tr>
<td>72</td>
<td>ACC$Q_LOGIN</td>
<td>quadword</td>
<td>Time in system format that process logged in</td>
</tr>
<tr>
<td>80</td>
<td>ACC$L_OWNER</td>
<td>longword</td>
<td>Process identification of owner</td>
</tr>
</tbody>
</table>

The length of the termination message is equated to the constant ACC$K_TERMLN.

3. All subprocesses created by a process must terminate before the creating process can be deleted. If subprocesses exist when their creator is deleted, they are automatically deleted.

For examples of subprocess creation, termination mailboxes, and system services that control the execution of processes, see Section 3.5, "Process Control Services."

4.19.1 Format of the Quota List

The system defines specific resources that are controlled by quotas. A quota limits the use of a particular system resource by a process.

The quota list addressed by the QUOTA argument of the $CREPRC system service consists of consecutive quota values contained in longwords, each preceded by a byte that indicates the quota type.

The $PQLDEF macro defines symbolic names for the quotas in the format:

PQL$_type

The quota list is terminated by the type code PQL$_LISTEND. For example, a quota list may be specified as:

```
QLIST: .BYTE PQL$_PRCLM  ; LIMIT NUMBER OF SUBPROCESSES
 .LONG 2                  ; MAX = 2 SUBPROCESSES
 .BYTE PQL$_ASTLM        ; LIMIT NUMBER OF ASTS
 .LONG 6                 ; MAX = 6 OUTSTANDING ASTS
 .BYTE PQL$_LISTEND      ; END OF QUOTA LIST
```

4.19.2 Quota Descriptions

The individual quota types are described below. Each description also indicates the following characteristics of the quota:

- Minimum value. A process cannot be created if it does not have a quota equal to or greater than this minimum.
- Default value. If the quota list does not specify a value for a particular quota, the system assigns the process this default value.

4-40
SYSTEM SERVICE DESCRIPTIONS
$CREPRC - CREATE PROCESS

- Deductible/Non-deductible. When a subprocess is created, the value specified for a deductible quota is subtracted from the current quota value of the creator. These quotas are returned to the creating process when the subprocess is deleted. Non-deductible quotas are not subtracted.

Quotas are never deducted from the creator when a detached process is created.

Note that the minimum and default values listed are not necessarily those provided at your installation; they are, however, the values recommended for general use.

Section 4.19.3 describes how these characteristics may affect quota assignments.

PQL$_ASTLM
AST limit. This quota restricts both the number of outstanding AST routines specified in system service calls that accept an AST address and the number of scheduled wakeup requests that can be issued.

Minimum: 2
Default: 10
Non-deductible

PQL$_BIOLM
Buffered I/O limit. This quota limits the number of outstanding system-buffered I/O operations. A buffered I/O operation is one which uses an intermediate buffer from the system pool rather than a buffer specified in a process's $QIO request.

Minimum: 2
Default: 6
Non-deductible

PQL$_BYTLM
Buffered I/O byte count quota. This quota limits the amount of system space that can be used to buffer I/O operations or to create temporary mailboxes.

Minimum: 1024
Default: 10240
Deductible

PQL$_CPULM
CPU time limit. This quota can be used to limit the total amount of CPU time used by a process. If the quota is specified as 0, there is no CPU time limit; the creating process, however, must have unlimited CPU time itself in order to grant the created process unlimited time.

If specified, the CPU time limit must be specified in units of 10 milliseconds. This quota is consumable; when the time limit has been used, the process is deleted. If a subprocess is given limited CPU time, the amount of time used is not returned to the creator when the subprocess is deleted.

Minimum: 0
Default: 0
Deductible
PQL$ DIOLM
Direct I/O quota. This quota limits the number of outstanding direct I/O operations. A direct I/O operation is one for which the system locks the pages containing the associated I/O buffer in memory for the duration of the I/O operation.

Minimum: 2
Default: 6
Non-deductible

PQL$ FillLM
Open file quota. This quota limits the number of files that a process can have open at one time.

Minimum: 2
Default: 20
Deductible

PQL$ PGFLQuota
Paging file quota. This quota limits the number of pages that can be used to provide secondary storage in the paging file for a process's execution.

Minimum: 256
Default: 10000
Deductible

PQL$ PRCLM
Subprocess quota. This quota limits the number of subprocesses a process can create.

Minimum: 0
Default: 8
Deductible

PQL$ TQELM
Timer queue entry quota. This quota limits both the number of timer queue requests a process can have outstanding and the creation of temporary common event flag clusters.

Minimum: 0
Default: 8
Deductible

PQL$ WSDEFAULT
Default working set size. This quota defines the number of pages in the default working set for any image executed by the process. The maximum size that can be specified for this quota is determined by the working set size quota.

Minimum: 10
Default: 100
Non-deductible

PQL$ WSQUOTA
Working set size quota. This quota limits the maximum size to which an image can expand its working set size with the Adjust Working Set Limit ($ADJWSL) system service.

Minimum: 10
Default: 200
Non-deductible
4.19.3 Quota Values

Values specified in the quota list are not necessarily the quotas that will actually be assigned to the created process. The $CREPRC system service performs the following steps to determine the quota values that will be assigned:

1. It constructs a default quota list for the process being created, assigning it the default values for all quotas.

2. It reads the specified quota list, if any, and updates the corresponding items in the default list. If the quota list contains multiple entries for a quota, only the last specification is used.

3. For each item in the updated quota list, it compares the quota value with the minimum value required for the quota and uses the larger value.

If a subprocess is being created:

1. The resulting value is compared with the current value of the corresponding quota of the creator. If the value exceeds the creator's quota, the status code $SS$ EXQUOTA is returned and the subprocess is not created.

2. If the quota is a deductible quota, it deducts the resulting value from the creator's quota and verifies that the creator will still have at least the minimum quota required. If not, the status code $SS$ EXQUOTA is returned and the subprocess is not created.

If a detached process is created, the resulting values are not compared with the creator's, nor are quotas deducted. Moreover, the service does not check that a specified quota value exceeds the maximum allowed by the system.
SCRETVA

4.20 SCRETVA - CREATE VIRTUAL ADDRESS SPACE

The Create Virtual Address Space system service adds a range of pages
to a process's virtual address space for the execution of the current
image.

Macro Format:

$SCRETVA inadr,[retadr],[acmode]

High-Level Language Format:

SYSSCRETVA(inadr,[retadr],[acmode])

inadr
address of a 2-longword array containing the starting and ending
virtual addresses of the pages to be created. If the starting
and ending virtual addresses are the same, a single page is
created. Only the virtual page number portion of the virtual
addresses is used; the low-order 9 bits are ignored.

retadr
address of a 2-longword array to receive the starting and ending
virtual addresses of the pages actually created.

acmode
access mode and protection for the new pages. The specified
access mode is maximized with the caller's access mode. The
protection of the pages is read/write for the resultant access
mode and those more privileged.

Return Status:

SS$NORMAL
Service successfully completed.

SS$ACCVIO
The input address array cannot be read, or the return address
array cannot be written, by the caller.

SS$EXQUOTA
The process has exceeded its paging file quota.

SS$INSFWSL
The process's working set limit is not large enough to
accommodate the increased size of the virtual address space.

SS$NOPRIV
A page in the specified range is in the system address space.

SS$PAGOWNVIO
A page in the specified range already exists and cannot be
deleted because it is owned by a more privileged access mode than
that of the caller.

SS$VASFULL
The process's virtual address space is full; no space is
available in the page tables for the requested pages.
SYSTEM SERVICE DESCRIPTIONS
$CRETVA - CREATE VIRTUAL ADDRESS SPACE

Resources Required/Returned:

The processes paging file quota (PGFQOUTA) and working set limit quota (WSQOUTA) must be sufficient to accommodate the increased size of the virtual address space.

Notes:

1. Pages are created starting at the address contained in the first longword of the location addressed by the parameter INADR and ending with the second longword. The ending address can be lower than the starting address. The return address array indicates the byte addresses of the pages created.

2. If an error occurs while creating pages, the return array, if requested, indicates the pages that were successfully created before the error occurred. If no pages were created, both longwords of the return address array contain a -1.

3. Each page to be created is first deleted, if necessary, and then reinitialized to a demand-zero page.

The Expand Program-Control Region ($EXPREG) also adds pages to a process's virtual address space. For additional details on page creation and deletion, see Section 3.8.2, "Increasing and Decreasing Virtual Address Space."
$CRMPSC

4.21 $CRMPSC — CREATE AND MAP SECTION

The Create and Map Section system service identifies a disk file for use as a global section or a private section and optionally makes the correspondence between virtual blocks in the file and pages in the caller's virtual address space. If the section already exists, the service maps it. Depending on the actual operation requested, certain arguments are required or optional. Table 4-1 summarizes how the $CRMPSC system service interprets the arguments passed to it, and under what circumstances it requires or ignores arguments.

Macro Format:

$CRMPSC [inadr],[retadr],[acmode],[flags],[gsdnam],[ident]
,[replag],[chan],[pagcnt],[vbn],[prot],[pfc]

High-Level Language Format:

SYSS$CRMPSC([inadr],[retadr],[acmode],[flags],[gsdnam],[ident]
,[replag],[chan],[pagcnt],[vbn],[prot],[pfc])

inadr
address of a 2-longword array containing the starting and ending virtual addresses in the process's virtual address space into which the section is to be mapped. If the starting and ending virtual addresses are the same, a single page is mapped. Only the virtual page number portion of the virtual addresses is used; the low-order 9 bits are ignored.

If this argument is not specified, or specified as 0, the section is not mapped.

retadr
address of a 2-longword array to receive the starting and ending virtual addresses of the pages into which the section was actually mapped.

acmode
access mode to be the owner of the pages created during the mapping. The access mode is maximized with the access mode of the caller.

flags
mask defining the section type and characteristics. Flag bit settings may be ORed together to override default attributes. The SSECDEF macro defines symbolic names for the flag bits in the mask. Their meanings, and the default values they override, are:

<table>
<thead>
<tr>
<th>Flag</th>
<th>Meaning</th>
<th>Default Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEC$M_GBL</td>
<td>Global section</td>
<td>Private section</td>
</tr>
<tr>
<td>SEC$M_CRF</td>
<td>Pages are copy-on-reference</td>
<td>Pages are shared</td>
</tr>
<tr>
<td>SEC$M_DZERO</td>
<td>Pages are demand-zero pages</td>
<td>Pages are not zeroed when copied</td>
</tr>
<tr>
<td>SEC$M_WRT</td>
<td>Read/write section</td>
<td>Read-only</td>
</tr>
<tr>
<td>SEC$M_PERM</td>
<td>Permanent</td>
<td>Temporary</td>
</tr>
<tr>
<td>SEC$M_SYSGBL</td>
<td>System global section</td>
<td>Group global section</td>
</tr>
</tbody>
</table>
gsdnam
address of a character string descriptor pointing to the 1- to
15-character text name string for the global section. For group
sections, the global section name is implicitly qualified
by the group number of the process creating the global section.

ident
address of a quadword indicating the version number of a global
section, and, for processes mapping to an existing global
section, the criteria for matching the identification.

The version number is in the second longword. The version number
contains two fields: a minor identification in the low-order 24
bits and a major identification in the high-order 8 bits. Values
for these fields can be assigned by installation convention to
differentiate versions of global sections. If no version number
is specified when a section is created, processes that specify a
version number when mapping cannot access the global section.

The first longword specifies, in its low-order 3 bits, the
matching criteria. The valid values, symbolic names by which
they can be specified, and their meanings are:

<table>
<thead>
<tr>
<th>Value/Name</th>
<th>Match Criteria</th>
</tr>
</thead>
<tbody>
<tr>
<td>0  SEC$K_MATALL</td>
<td>Match all versions of the section</td>
</tr>
<tr>
<td>1  SEC$K_MATEQU</td>
<td>Match only if major and minor identifications match</td>
</tr>
</tbody>
</table>
| 2  SEC$K_MATLEQ | Match if the major identifications are equal and the minor
                  identification of the mapper is
                  less than or equal to the minor
                  identification of the global section |

The match control field is ignored when a section is created. If
no address is specified, or is specified as 0 (the default), the
version number and match control fields default to 0.

replap
relative page number within the section of the first page in the
section to be mapped. If not specified, or specified as 0 (the
default), the global section is mapped beginning with the first
virtual block in the file.

chan
number of the channel on which the file has been accessed. The
file must have been accessed with an RMS $OPEN macro; the file
options parameter (POF) in the FAB must indicate a user file open
(UFO keyword). The access mode at which the channel was opened
must be the same or less privileged than the access mode of the
caller.

pagcnt
number of pages in the section. The specified page count is
compared with the number of pages in the section file; if they
are different, the lower value is used. If the page count is not
specified, or specified as 0 (the default) the size of the
section file is used.

vbn
virtual block number in the file that marks the beginning of the
section. If not specified, or specified as 0 (the default) the
section is created beginning with the first virtual block in the
file.
prot
numeric value representing the protection mask to be applied to
the global section.

The mask contains four 4-bit fields:

<table>
<thead>
<tr>
<th>15</th>
<th>11</th>
<th>7</th>
<th>3</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORLD</td>
<td>GROUP</td>
<td>OWNER</td>
<td>SYSTEM</td>
<td></td>
</tr>
</tbody>
</table>

Bits read from right to left in each field, when clear, indicate
that read, write, execute, and delete privileges, in that order,
are granted to the particular category of user.

Only read and write privileges are meaningful for global section
protection.

If not specified, or specified as 0, read and write privileges
are granted to all users.

pfc
page fault cluster size. If specified, the cluster size
indicates how many pages are to be brought into memory when a
page fault occurs for a single page.

Return Status:

SS$_NORMAL
Service successfully completed. The specified global section
already existed and has been mapped.

SS$_CREATED
Service successfully completed. The specified global section did
not previously exist and has been created.

SS$_ACCVIO
The input address array or the global section name or name
descriptor cannot be read, or the return address array cannot be
written, by the caller.

SS$_ENDOFFILE
Warning. The starting virtual block number specified is beyond
the logical end-of-file.

SS$_GPTFULL
There is no more room in the system global page table to set up
page table entries for the section.

SS$_GSDFULL
There is no more room in the system space allocated to maintain
control information for global sections.

SS$_EXQUOTA
The process exceeded its paging file quota while creating
copy-on-reference pages.

SS$_INSFWSL
The process's working set limit is not large enough to
accommodate the increased size of the address space.
## SYSTEM SERVICE DESCRIPTIONS
**$CRMPSC – CREATE AND MAP SECTION**

### Table 4-1
Arguments for the $CRMPSC System Service

<table>
<thead>
<tr>
<th>Argument</th>
<th>Create and Map Global Section</th>
<th>Map Global Section</th>
<th>Create and Map Private Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>INADR</td>
<td>Optional&lt;sup&gt;2&lt;/sup&gt;</td>
<td>Required</td>
<td>Required</td>
</tr>
<tr>
<td>RETADR</td>
<td>Optional</td>
<td>Optional</td>
<td>Optional</td>
</tr>
<tr>
<td>ACMODE</td>
<td>Optional</td>
<td>Optional</td>
<td>Optional</td>
</tr>
<tr>
<td>FLAGS</td>
<td>SEC$M GBL</td>
<td>Required</td>
<td>Ignored</td>
</tr>
<tr>
<td></td>
<td>SEC$M CRF</td>
<td>Optional</td>
<td>Not used</td>
</tr>
<tr>
<td></td>
<td>SEC$M_DZRO</td>
<td>Optional</td>
<td>Not used</td>
</tr>
<tr>
<td></td>
<td>SEC$M_WRT</td>
<td>Optional&lt;sup&gt;2&lt;/sup&gt;</td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>SEC$M_PERM</td>
<td>Optional&lt;sup&gt;1&lt;/sup&gt;</td>
<td>Not used</td>
</tr>
<tr>
<td></td>
<td>SEC$M_SYSGBL</td>
<td>Optional</td>
<td>Not used</td>
</tr>
<tr>
<td>GSDNAM</td>
<td>Required</td>
<td>Required</td>
<td>Not used</td>
</tr>
<tr>
<td>IDENT</td>
<td>Optional</td>
<td>Optional</td>
<td>Not used</td>
</tr>
<tr>
<td>RELPAG</td>
<td>Optional</td>
<td>Optional</td>
<td>Not used</td>
</tr>
<tr>
<td>CHAN</td>
<td>Required</td>
<td></td>
<td>Required</td>
</tr>
<tr>
<td>PAGCNT</td>
<td>Required</td>
<td></td>
<td>Required</td>
</tr>
<tr>
<td>VBN</td>
<td>Optional</td>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td>PROT</td>
<td>Optional</td>
<td></td>
<td>Not used</td>
</tr>
<tr>
<td>PFC</td>
<td>Optional</td>
<td></td>
<td>Optional</td>
</tr>
</tbody>
</table>

<sup>1</sup> The Map Global Section ($MGBLSC) system service maps an existing global section.

<sup>2</sup> If the $CRMPSC system service is called to create, but not map, a global section, the global section must be permanent.

### $S$S$S$ _IVCHAN_
- An invalid channel number was specified, that is a channel number of 0 or a number larger than the number of channels available.

### $S$S$S$ _IVCHNLSEC_
- The channel number specified is currently active.

### $S$S$S$ _IVLOGNAM_
- The specified global section name has a length of 0, or has more than 15 characters.

### $S$S$S$ _IVSECPLG_
- An invalid flag has been specified. Either a reserved flag has been set, or one requiring a user privilege.

### $S$S$S$ _IVSECIDCFL_
- The match control field of the global section identification is invalid.

4-49
SS$ _NOPRIV
The process does not have the privilege to create a system global
section or a permanent group global section.
A page in the input address range is in the system address space.
The specified channel does not exist or was assigned from a more
privileged access mode.

SS$ _PAGOWNVIO
A page in the specified input address range is owned by a more
privileged access mode.

SS$ _SECTBLFUL
There are no entries available in the system global section
table.

SS$ _VASFULL
The process's virtual address space is full; no space is
available in the page tables for the pages created to contain the
mapped global section.

Privilege Restrictions:
The user privilege SYSGBL is required to create a system global
section; the PRMGBL privilege is required to create a permanent
global section.

Resources Required/Returned:
The process's working set limit quota (WSQUOTA) must be
sufficient to accommodate the increased size of the virtual
address space when mapping a section. If the section pages are
copy-on-reference, the process must also have sufficient paging
file quota (PGFLQUOTA).

Notes:
1. When the $CRMPSC system service maps a section, it calls the
   Create Virtual Address Space ($CRETVA) system service to add
   the pages specified by the INADR argument to the process's
   virtual address space. The specified virtual addresses can
   be in the program (P0) region or the control (P1) region.

   If a global section is of an unknown size, a process can obtain the virtual address of the first available page in its
program or control region from the Get Job/Process
Information ($GETJPI) system service and use the address
returned as the starting address in the input address array.
The ending address may be a very high address (if the section is
to be mapped in the program region) or a very low address
(if mapped in the control region). The $CRMPSC system
service returns the virtual addresses of the pages created in
the RETADR argument, if specified. The section is mapped
from a low address to a high address, regardless of whether
the section is mapped in the program or control region.
2. If an error occurs during the mapping of a global section, the return address array, if specified, indicates the pages that were successfully mapped when the error occurred. If no pages were mapped, both longwords of the return address array contain -1.

If the global section is permanent, it is not deleted if the mapping operation fails.

For examples of the creation and mapping of private and global sections, see Section 3.8.6, "Sections."
$DACEFC

4.22 $DACEFC – DISASSOCIATE COMMON EVENT FLAG CLUSTER

The Disassociate Common Event Flag Cluster system service releases the calling process's association with a common event flag cluster.

Macro Format:

$DACEFC efn

High-Level Language Format:

SYS$DACEFC(efn)

efn
number of any event flag in the common cluster to be disassociated. The flag number must be in the range of 64 through 95 for cluster 2 and 96 through 127 for cluster 3.

Return Status:

SS$ NORMAL
Service successfully completed.

SS$ ILLEFC
An illegal event flag number was specified. The number must be in the range of event flags 64 through 127.

Notes:

1. The count of processes associated with the cluster is decreased for each process that disassociates. When the image that associated with a cluster exits, the system performs an implicit disassociate for the cluster. When the count of processes associated with a temporary cluster or a permanent cluster marked for deletion reaches zero, the cluster is automatically deleted.

2. If a process issues this service specifying an event flag cluster with which it is not associated, the service completes successfully.

For an example of the $DACEFC system service and a description of the creation and association of common event flag clusters, see Section 3.1.4, "Common Event Flag Clusters."
4.23 $DALLOC - DEALLOCATE DEVICE

The Deallocate Device system service deallocates a previously allocated device. Exclusive use by the issuing process is relinquished and other processes can assign or allocate the device.

Macro Format:

$DALLOC [devnam] , [acmode]

High-Level Language Format:

SYS$DALLOC([devnam] , [acmode])

devnam
address of a character string descriptor pointing to the device name string. The string may be either a physical device name or a logical name. If the first character in the string is an underline character (_), the name is considered a physical device name. Otherwise, a single level of logical name translation is performed and the equivalence name, if any, is used. The final name, however, cannot contain a node name unless the name is that of the host system.

If no device name is specified, all devices allocated by the process from access modes equal to or less privileged than that specified are deallocated.

acmode
access mode on behalf of which the deallocation is to be performed. The access mode is maximized with the access mode of the caller.

Return Status:

SS$ NORMAL
Service successfully completed.

SS$ ACCVIO
The device name string or string descriptor cannot be read by the caller.

SS$ DEVASSIGN
Warning. The device cannot be deallocated because the process still has channels assigned to it.

SS$ DEVNOTALLOC
Warning. The device is not allocated to the requesting process.

SS$ IVDEVNAM
No device name string was specified or the device name string contains invalid characters.

SS$ IVLOGNAM
The device name string has a length of 0 or has more than 63 characters.
SYSTEM SERVICE DESCRIPTIONS
$DALLOCS - DEALLOCATE DEVICE

SS$ NOPRIV
   - The device was allocated from a more privileged access mode.

SS$ NOSUCHDEV
   - Warning. The specified device does not exist in the host system.

Privilege Restrictions:

   An allocated device can be deallocated only from access modes equal to or more privileged than the access mode from which the original allocation was made.

Notes:

   1. A process cannot deallocate a device at any time. If, at the time of deallocation, the issuing process has one or more I/O channels assigned to the device, the device remains allocated.

   2. The system automatically deallocates all devices that were allocated at user mode at image exit.

   3. If an attempt is made to deallocate a mailbox, success is returned but no operation is performed.

For an example of how to use this service and additional notes on device allocation, see Section 3.4.9, "Device Allocation."
4.24 $DASSGN - DEASSIGN I/O CHANNEL

The Deassign I/O Channel system service releases an I/O channel acquired for input/output operations with the Assign I/O Channel ($ASSIGN) system service.

Macro Format:

$DASSGN  chan

High-Level Language Format:

SYS$DASSGN(chan)

chan
number of the I/O channel to be deassigned.

Return Status:

SS$ NORMAL
Service successfully completed.

SS$_IVCHAN
An invalid channel number was specified; that is, a channel number of 0 or a number larger than the number of channels available.

SS$_NOPRIV
The specified channel is not assigned, or was assigned from a more privileged access mode.

Privilege Restrictions:

An I/O channel can be deassigned only from an access mode equal to or more privileged than the access mode from which the original channel assignment was made.

Notes:

1. When a channel is deassigned, any outstanding I/O requests on the channel are canceled. If a file is open on the specified channel, the file is closed.

2. If a mailbox was associated with the device when the channel was assigned, the linkage to the mailbox is cleared if there are no more channels assigned to the mailbox.

3. If the I/O channel was assigned for a network operation, the network link is disconnected. For more information on channel assignment and deassignment for network operations, see the DECnet-VAX User's Guide.
4. If the specified channel is the last channel assigned to a device that has been marked for dismounting, the device is dismounted.

5. I/O channels are automatically deassigned at image exit.

For an example of the $DASSGN system service and additional information on channel assignment, see Section 3.4.1, "Assigning Channels."
4.25 $DCLAST - DECLARE AST

The Declare AST system service queues an AST for the calling or for a less privileged access mode. For example, a routine executing in supervisor mode can declare an AST for either supervisor or user mode.

Macro Format:

$DCLAST astadr,[astprm],[acmode]

High-Level Language Format:

SYS$DCLAST(astadr,[astprm],[acmode])

astadr
address of the entry mask of the AST service routine.

astprm
value to be passed to the AST routine as an argument, if any.

acmode
access mode for which the AST is to be declared. This access mode is maximized with the access mode of the caller. The resultant mode is the access mode for which the AST is declared.

Return Status:

SS$ NORMAL
Service successfully completed.

SS$ EXQUOTA
The process has exceeded its AST limit quota.

SS$ INSFMEM
Insufficient system dynamic memory is available to complete the service and the process has disabled resource wait mode with the Set Resource Wait Mode ($SETRWM) system service.

Resources Required/Returned:

1. The Declare AST system service requires system dynamic memory.

2. This service uses the process's AST limit quota (ASTLM).

Note:

The $DCLAST system service does not validate the address of the AST service routine. If an illegal address, for example, an address of 0, is specified, an access violation occurs when the AST service routine is given control.

For an example of the $DCLAST system service and notes and coding conventions for AST service routines, see Section 3.2, "Asynchronous System Trap (AST) Services."
$DCLCMH

4.26 $DCLCMH - DECLARE CHANGE MODE OR COMPATIBILITY MODE HANDLER

Declare Change Mode or Compatibility Mode Handler ($DCLCMH) system service establishes the address of a routine to receive control when (1) a Change Mode to User or Change Mode to Supervisor instruction trap occurs, or (2) a compatibility mode fault occurs.

Macro Format:

$DCLCMH address , [prvhn] , [type]

High-Level Language Format:

SYS$DCLCMH(address , [prvhn] , [type])

address
address of a routine to receive control when a change mode trap or a compatibility mode fault occurs. An address of 0 clears a previously declared handler.

prvhn
address of a longword to receive the address of a previously declared handler.

type
type indicator. If specified as 0 (the default), a change mode handler is declared for the access mode at which the request is issued. If specified as 1, a compatibility mode handler is declared.

Return Status:

SS$NORMAL
Service successfully completed.

SS$ACCVI0
The longword to receive the address of the previous change mode handler cannot be written by the caller.

Notes:

1. A change mode handler provides users with a dispatching mechanism similar to that used for system service calls. It allows a routine that executes in supervisor mode to be called from user mode. The change mode handler is declared from supervisor mode; when the process is then executing in user mode and issues a Change Mode to Supervisor instruction, the change mode handler receives control, and executes in supervisor mode.

2. Compatibility mode handlers are used by the operating system to bypass normal condition handling procedures when an image executing in compatibility mode incurs a compatibility mode exception.
3. When the change mode or compatibility mode handler receives control, the stack pointer points to the change mode code specified in the change mode instruction or the compatibility exception type code. On exit, the handler must remove the code from the stack, then relinquish control with an REI instruction.

4. A change mode handler can be declared only from user or supervisor modes.
$DCLEXH

4.27 $DCLEXH - DECLARE EXIT HANDLER

The Declare Exit Handler system service describes an exit handling routine to receive control when an image exits. Image exit normally occurs when the image currently executing in a process returns control to the operating system. Image exit may also occur when the Exit ($EXIT) or Force Exit ($FORCEX) system services are called.

Macro Format:

$DCLEXH desblk

High-Level Language Format:

SYS$DCLEXH(desblk)

desblk
   address of a control block describing the exit handler. The exit control block has the format:

```
<table>
<thead>
<tr>
<th>31</th>
<th>8 7</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>exit handler address</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>u</td>
</tr>
<tr>
<td></td>
<td></td>
<td>n</td>
</tr>
<tr>
<td></td>
<td>address to store reason for exit</td>
<td></td>
</tr>
<tr>
<td></td>
<td>additional arguments for exit handler, if any</td>
<td></td>
</tr>
</tbody>
</table>
```

The system fills in the first longword.

Return Status:

SS$_NORMAL
   Service successfully completed.

SS$_ACCVIO
   The first longword of the exit control block cannot be written by the caller.

SS$_NOHANDLER
   Warning. No exit handler control block address was specified; or, the address specified is 0.
Notes:

1. Exit handlers are described by exit control blocks. The operating system maintains a separate list of these control blocks for user, supervisor, and executive modes. The $DCELEXH system service adds the description of an exit handler to the front of one of these lists. The actual list to which the exit control block is added is determined by the access mode of the caller.

   This service can only be called from user, supervisor, and executive modes.

2. At image exit, the exit handlers declared from user mode are called first; they are called in the reverse order from which they were declared.

   Each exit handler is executed only once; it must be redeclared before it can be executed again. The exit handling routine is called as a normal procedure with the argument list specified in the 3rd through nth longwords of the exit control block. The first argument is the address of a longword to receive a system status code indicating the reason for exit; the system always fills in this longword before calling the exit handler.

3. The Cancel Exit Handler ($SCANEXH) removes an exit control block from the list.

For an example of an exit control block and a description of the action the system takes when an image exits, see Section 3.5.6, "Image Exit."
$DELOG

4.28 $DELOG - DELETE LOGICAL NAME

The Delete Logical Name system service deletes a logical name and its equivalence name from the process, group, or system logical name table.

Macro Format:

$DELOG [tblflg],[lognam],[acmode]

High-Level Language Format:

SYS$DELOG([tblflg],[lognam],[acmode])

tblflg:
logical name table number. A value of 0 (the default) indicates the system table, 1 indicates the group table, and 2 indicates the process table.

lognam:
address of a character string descriptor pointing to the logical name string. If omitted, all logical names the process is privileged to delete in the specified table are deleted.

acmode:
access mode associated with the process logical name table entry. The specified access mode is maximized with the access mode of the caller; only the logical name entered at the resulting access mode is deleted. This argument is used only for deleting names from the process logical name table.

Return Status:

SS$ NORMAL
- Service successfully completed.

SS$ ACCVIO
- The logical name string or string descriptor cannot be read by the caller.

SS$ IVLOGNAM
- The logical name string has a length of 0, or has more than 63 characters.

SS$ IVLOGTAB
- An invalid logical name table number was specified.

SS$ NOLOGNAM
- Either (1) the specified logical name does not exist in the specified logical name table, or (2) the specified logical name exists in the process logical name table but the entry was made from a more privileged access mode.

SS$ NOPRIV
- The process does not have the privilege to delete an entry from the specified logical name table.
SYSTEM SERVICE DESCRIPTIONS
$DELLOG - DELETE LOGICAL NAME

Privilege Restrictions:

   The user privileges GRPNAM and SYSNAM are required to delete
   names from the group and system logical name tables, respectively.

Resources Required/Returned:

   1. Deletion of a logical name from the group or system table
      returns storage to system dynamic memory.

   2. When a logical name is deleted from the process logical name
      table, the number of bytes in the control region of the
      process's virtual address space required to maintain the
      table entry become available for other process logical name
      table entries.

Notes:

   1. Logical names can be deleted from the command stream with the
      DEASSIGN command.

   2. Names in the process logical name table that are qualified by
      user mode are automatically deleted at image exit.

For an example of the $DELLOG system service and additional details on
logical name creation and translation, see Section 3.3, "Logical Name
Services."
$DELMBX

4.29 $DELMBX - DELETE MAILBOX

The Delete Mailbox system service marks a permanent mailbox for deletion. The actual deletion of the mailbox and of its associated logical name assignment occur when no more I/O channels are assigned to the mailbox.

Macro Format:

$DELMBX chan

High-Level Language Format:

SYS$DELMBX(chan)

chan
number of the channel assigned to the mailbox.

Return Status:

SS$NORMAL
Service successfully completed.

SS$DEVNOTMBX
The specified channel is not assigned to a mailbox.

SS$IVCHAN
An invalid channel number was specified, that is, a channel number of 0 or a number larger than the number of channels available.

SS$NOPRIV
The specified channel is not assigned to a device, the process does not have the privilege to delete a permanent mailbox, or the access mode of the caller is less privileged than the access mode from which the channel was assigned.

Privilege Restrictions:

1. The user privilege PRMMBX is required to delete a permanent mailbox.

2. A mailbox can be deleted only from an access mode equal to or more privileged than the access mode from which the mailbox channel was assigned.
SYSTEM SERVICE DESCRIPTIONS
$DELMBX - DELETE MAILBOX

Notes:

1. Temporary mailboxes are automatically deleted when their reference count goes to zero.

2. The $DELMBX system service does not deassign the channel assigned by the caller, if any. The caller must deassign the channel with the Deassign I/O Channel ($DASSGN) system service.

For information on the creation and use of mailboxes, see Section 3.4.13, "Mailboxes."
$DELPRC

4.30 $DELPRC - DELETE PROCESS

The Delete Process system service allows a process to delete itself or another process.

Macro Format:

$DELPRC [pidadr] ,[prcnam]

High-Level Language Format:

SYS$DELPRC([pidadr] ,[prcnam])

pidadr
address of a longword containing the process identification of the process to be deleted.

prcnam
address of a character string descriptor pointing to the process name string. The process name is implicitly qualified by the group number of the process issuing the delete.

If neither a process identification nor a process name is specified, the caller is deleted and control is not returned. For details on how the service interprets the PIDADR and PRCNAM arguments, see Table 3-3. Table 3-3 is in Section 3.5, "Process Control Services."

Return Status:

SS$ NORMAL
  Service successfully completed.

SS$ ACCVIO
  The process name string or string descriptor cannot be read, or the process identification cannot be written, by the caller.

SS$ INSFMEM
  Insufficient system dynamic memory is available to complete the service and the process has disabled resource wait mode with the Set Resource Wait Mode ($SETRWM) system service.

SS$ NONEXPR
  Warning. The specified process does not exist, or an invalid process identification was specified.

SS$ NOPRIV
  The process does not have the privilege to delete the specified process.
Privilege Restrictions:

User privileges are required to delete:

- Other processes in the same group (GROUP privilege)
- Any process in the system (WORLD privilege)

Resources Required/Returned:

1. The Delete Process system service requires system dynamic memory.

2. Deductible resource quotas granted to subprocesses are returned to the creator when the subprocesses are deleted.

Notes:

1. When a subprocess is deleted, a termination message is sent to its creator, provided that the mailbox to receive the message still exists and the creating process has access to the mailbox. The termination message is sent before the final rundown is initiated; thus, the creator may receive the message before the process deletion is complete.

2. Due to the complexity of the required rundown operations, a significant time interval occurs between a delete request and the actual disappearance of the process. The Delete Process service, however, returns immediately after initiating the rundown operation. If subsequent delete requests are issued for a process currently being deleted, the requests return immediately with a return status code indicating successful completion.

For a complete list of the actions performed by the system when it deletes a process, see Sections 3.5.6, "Image Exit" and 3.5.7, "Process Deletion."
$DELTVA

4.31 $DELTVA - DELETE VIRTUAL ADDRESS SPACE

The Delete Virtual Address Space system service deletes a range of addresses from a process's virtual address space. Upon successful completion of the service, the deleted pages are inaccessible; any references to them cause access violations.

Macro Format:

$DELTVA inadr ,[retadr] , [acmode]

High-Level Language Format:

SYS$DELTVA(inadr ,[retadr] , [acmode])

inadr
address of a 2-longword array containing the starting and ending virtual addresses of the pages to be deleted. If the starting and ending virtual addresses are the same, a single page is deleted. Only the virtual page number portion of the virtual addresses is used; the low-order 9 bits are ignored.

retadr
address of a 2-longword array to receive the starting and ending virtual addresses of the pages actually deleted.

acmode
access mode on behalf of which the service is to be performed. The specified access mode is maximized with the access mode of the caller. The resultant access mode is used to determine whether the caller can actually delete the specified pages.

Return Status:

SS$ NORMAL
Service successfully completed.

SS$ ACCVIO
The input address array cannot be read, or the return address array cannot be written, by the caller.

SS$ NOPRIV
A page in the specified range is in the system address space.

SS$ PAGOWNVIO
A page in the specified range is owned by an access mode more privileged than the access mode of the caller.

Privilege Restrictions:

Pages can only be deleted from access modes equal to or more privileged than the access mode of the owner of the pages.
Notes:

1. The $DELTVA system service deletes pages starting at the address contained in the second longword of the INADR array and ending at the address in the first longword. Thus, if the same address array is used for the Create Virtual Address Space ($CRETVA) as for the $DELTVA system service, the pages are deleted in the reverse order from which they were created.

2. If any of the pages in the specified range have already been deleted or do not exist, the service continues as if the pages were successfully deleted.

3. If an error occurs while deleting pages, the return array, if requested, indicates the pages that were successfully deleted before the error occurred. If no pages are deleted, both longwords in the return address array contain a -1.

For an example of the $DELTVA system service and additional information on page creation and deletion, see Section 3.8.2, "Increasing and Decreasing Virtual Address Space."
$DGBLSC

4.32 $DGBLSC — DELETE GLOBAL SECTION

The Delete Global Section system service marks an existing permanent global section for deletion. The actual deletion of the global section takes place when all processes that have mapped the global section have deleted the mapped pages.

Macro Format:

$DGBLSC [flags] ,gsdnam ,[ident]

High-Level Language Format:

SYS$DGBLSC([flags] ,gsdnam ,[ident])

flags

mask indicating global section characteristics. The only significant bit used for the deletion of global sections is the group/system flag. If this argument is specified as 0 (the default), it indicates that the global section is a group global section; if specified as SEC$M_SYSGBL, it indicates a system global section.

gsdnam

address of a character string descriptor pointing to the 1- to 15-character text name of the global section to be deleted. For group global sections, the global section name is implicitly qualified by the group number of the caller.

ident

address of a quadword indicating the version number of the global section to delete and the matching criteria to be applied.

The version number is in the second longword. The version number contains two fields: a minor identification in the low-order 24 bits and a major identification in the high-order 8 bits.

The first longword specifies, in the low-order 3 bits, the matching criteria. The valid values, symbolic names by which they can be specified, and their meanings are listed below:

<table>
<thead>
<tr>
<th>Value/Name</th>
<th>Match Criteria</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 SEC$K_MATALL</td>
<td>Match all versions of the section</td>
</tr>
<tr>
<td>1 SEC$K_MATEQU</td>
<td>Match only if major and minor identifications match</td>
</tr>
<tr>
<td>2 SEC$K_MATLEQ</td>
<td>Match if the major identifications are equal and the minor identification of the mapper is less than or equal to the minor identification of the global section.</td>
</tr>
</tbody>
</table>

If no address is specified, or is specified as 0 (the default), the version number and match control fields default to 0.
SYSTEM SERVICE DESCRIPTIONS
$DGBLSC - DELETE GLOBAL SECTION

Return Status:

SS$ NORMAL - Service successfully completed.

SS$ ACCVIO - The global section name or name descriptor or the section identification field cannot be read by the caller.

SS$ IVLOGNAM - The global section name has a length of 0, or has more than 15 characters.

SS$ IVSECFLLG - An invalid flag has been specified. Either a reserved flag has been set, or one requiring a user privilege.

SS$ IVSECIDCFL - The section identification match control field is invalid.

SS$ NOPRIIV - The caller does not have the privilege to delete a system global section, or does not have read/write access to a group global section.

SS$ NOSUCHSEC - Warning. The specified global section does not exist.

Privilege Restrictions:

The user privileges SYSGBL and PRMGBL are required to delete system and permanent global sections, respectively.

Notes:

1. After a global section has been marked for deletion, any process that attempts to map it receives the warning return status code SS$ NOSUCHSEC.

2. Temporary global sections are automatically deleted when the count of processes using the section goes to 0.

3. This service does not unmmap a section from a process's virtual address space. When a process no longer requires use of a section, it can unmmap the section by calling the Delete Virtual Address Space ($DELTVA) system service to delete the pages to which the section is mapped.

For information on the creation and use of sections, see Section 3.8.6, "Sections."
$DLCEFC

4.33 $DLCEFC - DELETE COMMON EVENT FLAG CLUSTER

The Delete Common Event Flag Cluster system service marks a permanent common event flag cluster for deletion. The cluster is actually deleted when no more processes are associated with it.

Macro Format:

$DLCEFC  name

High-Level Language Format:

SYS$DLCEFC(name)

name  address of a character string descriptor pointing to the 1- to 15-character text name of the cluster. The name is implicitly qualified by the group number of the caller.

Return Status:

SS$ NORMAL
- Service successfully completed.

SS$ IVLOGNAM
- The cluster name string has a length of 0, or has more than 15 characters.

SS$ NOPRIV
- The process does not have the privilege to delete a permanent common event flag cluster.

Privilege Restrictions:

The user privilege PRMCEB is required to delete permanent common event flag clusters.

Notes:

1. The $DLCEFC system service does not perform an implicit disassociate request for the caller. A process disassociates with a cluster by calling the Disassociate Common Event Flag Cluster ($DACEFC) system service. The system performs an implicit disassociate for the cluster at image exit.

2. If the cluster has already been deleted or does not exist, the $DLCEFC service returns the status code SS$ NORMAL.

For an example of creating and using a common event flag cluster, see Section 3.1.4, "Common Event Flag Clusters."
4.34 $EXIT - EXIT

The Exit system service is used by the operating system to initiate image rundown when the current image in a process completes execution. Control normally returns to the command interpreter.

Macro Format:

$EXIT [code]

High-Level Language Format:

SYS$EXIT([code])

code
longword value to be saved in the process header as the completion status of the current image. If not specified in a macro call, a value of 1 is passed as the completion code. This value can be tested at the command level to provide conditional command execution.

Return Status:

No status codes are returned by this service because control is not returned to the caller; rather, an exit to the command interpreter is performed.

Note:

For a complete list of the actions taken by the system at image exit, see Section 3.5.6, "Image Exit."
SYSTEM SERVICE DESCRIPTIONS

$EXPREG

4.35  $EXPREG - EXPAND PROGRAM/CONTROL REGION

The Expand Program/Control Region system service adds a specified number of new virtual pages to a process's program region or control region for the execution of the current image. Expansion occurs at the current end of that region's virtual address space.

Macro Format:

$EXPREG  pagcnt ,[retadr] ,[acmode] ,[region]

High-Level Language Format:

SYSS$EXPREG(pagcnt ,[retadr] ,[acmode] ,[region])

pagcnt
number of pages to add to the current end of the program or control region.

retadr
address of a 2-longword array to receive the starting and ending virtual addresses of the pages actually added.

acmode
access mode and protection for the new pages. The specified access mode is maximized with the access mode of the caller. The protection of the pages is read/write for the specified access mode and more privileged access modes.

region
region indicator. A value of 0 (the default) indicates that the program region (P0 region) is to be expanded. A value of 1 indicates that the control region (P1 region) is to be expanded.

Return Status:

SS$_NORMAL
Service successfully completed.

SS$_ACCVIO
The return address array cannot be written by the caller.

SS$_EXQUOTA
The process exceeded its paging file quota.

SS$_ILLPAGCNT
The specified page count was less than 1.

SS$_INSFWSL
The process's working set limit is not large enough to accommodate the increased virtual address space.

SS$_VASFULL
The process's virtual address space is full; no space is available in the process page table for the requested regions.
SYSTEM SERVICE DESCRIPTIONS
$EXPREG - EXPAND PROGRAM/CONTROL REGION

Resources Required/Returned:

The process's paging file quota (PGFLQUOTA) and working set limit quota (WSQUOTA) must be sufficient to accommodate the increased size of the virtual address space.

Notes:

1. The new pages, which were previously inaccessible to the process, are created as demand-zero pages.

2. Because the bottom of the user stack is normally located at the end of the control region, expanding the control region is equivalent to expanding the user stack. The effect is to increase the available stack space by the specified number of pages.

3. The starting address returned is always the first available page in the designated region; therefore, the ending address is smaller than the starting address when the control region is expanded and is larger than the starting address when the program region is expanded.

4. If an error occurs while adding pages, the return address array, if requested, indicates the pages that were successfully added before the error occurred. If no pages were added, both longwords of the return address array contain a -1.

5. The information returned in the location addressed by the RETADR argument, if specified, can be used as the input range to the Delete Virtual Address Space ($DELTVA) system service. Pages can also be deleted with the Contract Program/Control Region ($CNTREG) system service.

For an example of the $EXPREG system service and additional information on creating and deleting pages, see Section 3.8.2, "Increasing and Decreasing Virtual Address Space."

Use LIB$... rather to expand
$FAO

4.36 $FAO - FORMATTED ASCII OUTPUT

The Formatted ASCII Output system service converts binary values into ASCII characters and returns the converted characters in an output string. It can be used to:

- Insert variable character string data into an output string
- Convert binary values into the ASCII representations of their decimal, hexadecimal, or octal equivalents and substitute the results into an output string.

Sections 4.36.2 through 4.36.5 provide syntactical notes, lists of valid FAO directives and parameters, and examples of using FAO.

Macro Format:

\[ $FAO \text{ ctrstr ,}[\text{outlen} \text{ ,outbuf },[\text{pl }],\text{[p2] }...,[\text{pn}]) \]

High-Level Language Format:

\[ \text{SYS$FAO}(\text{ctrstr },[\text{outlen} \text{ ,outbuf },[\text{pl }],\text{[p2] }...,[\text{pn}])) \]

ctrstr
address of a character string descriptor pointing to the control string. The control string consists of the fixed text of the output string and FAO directives.

outlen
address of a word to receive the actual length of the output string returned.

outbuf
address of a character string descriptor pointing to the output buffer. The fully formatted output string is returned in this buffer.

pl - pn
directive parameters contained in longwords. Depending on the directive, a parameter may be a value that is to be converted, the address of the string that is to be inserted, or a length or argument count. Each directive in the control string may require a corresponding parameter or parameters.

Return Status:

\[ \text{SYS$ BUFFEROVF} \]
Service successfully completed. The formatted output string overflowed the output buffer and has been truncated.

\[ \text{SYS$ NORMAL} \]
Service successfully completed.

\[ \text{SYS$ BADPARAM} \]
An invalid directive was specified in the FAO control string.
Notes:

1. The $FAO_S macro form uses a PUSHL instruction for all parameters (P1 through Pn) coded on the macro instruction; if a symbolic address is specified, it must be preceded with a pound sign (#) character or loaded into a register.

2. A maximum of 20 parameters can be specified on the $FAO macro instruction. If more than 20 parameters are required, use the $FAOL macro.

3. The $FAO system service executes at the access mode of the caller and does not check whether address arguments are accessible before it executes. Therefore, an access violation causes an exception condition if an input field cannot be read or an output field cannot be written. Note that an access violation can occur if an invalid length is specified for an argument, or if an FAO parameter is coded incorrectly.

4.36.1 $FAO_L - Formatted ASCII Output with List Parameter

The Formatted ASCII Output with List Parameter macro provides an alternate way to specify input parameters for a call to the $FAO system service.

Macro Format:

$FAOL ctrstr ,[outlen] ,outbuf ,prmlst

High-Level Language Format:

SYS$FAOL(ctrstr ,[outlen] ,outbuf ,prmlst)

ctrstr
address of a character string descriptor pointing to the control string. The control string consists of the fixed text of the output string and conversion directives.

outlen
address of a word to receive the actual length of the output string returned.

outbuf
address of a character string descriptor pointing to the output buffer. The fully formatted output string is returned in this buffer.

prmlst
address of the parameter list of longwords to be used as Pl through Pn.

The parameter list may be a data structure that already exists in a program and from which certain values are to be extracted.

Return Status:

Same as for $FAO system service.
4.36.2 FAO Directives

An FAO directive has the format:

!DD

! (exclamation mark) indicates that the following character or characters are to be interpreted as an FAO directive.

DD is a 1- or 2-character code indicating the action that FAO is to perform. Each directive may require one or more input parameters on the call to $FAO. All directive codes for FAO must be specified in uppercase letters.

Optionally, a directive may include:

- A repeat count
- An output field length

A repeat count is coded as follows:

!n(DD)

where n is a decimal value instructing FAO to repeat the directive for the specified number of parameters.

An output field length is specified as follows:

!lengthDD

where "length" is a decimal value instructing FAO to place the output resulting from a directive into a field of "length" characters in the output string.

A directive may contain both a repeat count and an output length, as shown below:

!n(lengthDD)

Repeat counts and output field lengths may be specified as variables, by using a # (number sign) in place of an absolute numeric value. If a # is specified for a repeat count, the next parameter passed to FAO must contain the count. If a # is specified for an output field length, the next parameter must contain the length value.

If a variable output field length is specified with a repeat count, only one length parameter is required; each output string will have the specified length.

4.36.3 FAO Control String and Parameter Processing

An FAO control string may be any length and may contain any number of FAO directives. The only restriction is on the use of the ! character (ASCII code `X21) in the control string. If a literal ! is required in the output string, the directive !! provides an escape.
When FAO processes a control string, each character that is not part of a directive is written into the output buffer. When a directive is encountered, it is validated; if it is not a valid directive, FAO terminates and returns an error status code. If the directive is valid, and if it requires one or more parameters, the next consecutive parameters specified are analyzed and processed.

FAO reads parameters from the argument list; it does not check the number of arguments it has been passed. If there are not enough parameters coded in the argument list, FAO will continue reading past the end of the list. It is your responsibility, when coding a call to $FAO, to ensure that there are enough parameters to satisfy the requirements of all the directives in the control string.

4.36.4 Summary of FAO Directives and Output Field Length Defaults

Table 4-2 summarizes the FAO directives, and lists the parameter(s) required by each directive. Table 4-3 summarizes how FAO determines the length of each output field in the control string as it processes directives and substitutes character strings in the control string while formatting the output buffer.

Examples in Section 4.36.5 describe in more detail how to use FAO.
### Table 4-2
**Summary of FAO Directives**

<table>
<thead>
<tr>
<th>Directive</th>
<th>Function</th>
<th>Parameter(s)(^1)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Character String Substitution</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IAC</td>
<td>Inserts a counted ASCII string</td>
<td>Address of the string; the first byte must contain the length</td>
</tr>
<tr>
<td>IAD</td>
<td>Inserts an ASCII string</td>
<td>1) Length of string 2) Address of string</td>
</tr>
<tr>
<td>IAF</td>
<td>Inserts an ASCII string; Replaces all nonprintable ASCII codes with periods (.)</td>
<td>1) Length of string 2) Address of string</td>
</tr>
<tr>
<td>IAS</td>
<td>Inserts an ASCII string</td>
<td>Address of quadword character string descriptor pointing to the string</td>
</tr>
<tr>
<td><strong>Numeric Conversion (zero-filled)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IOB</td>
<td>Converts a byte to octal</td>
<td>Value to be converted to ASCII representation</td>
</tr>
<tr>
<td>IOW</td>
<td>Converts a word to octal</td>
<td>For byte or word conversion, FAO uses only the low-order byte or word of the longword parameter</td>
</tr>
<tr>
<td>IOL</td>
<td>Converts a longword to octal</td>
<td></td>
</tr>
<tr>
<td>IXB</td>
<td>Converts a byte to hexadecimal</td>
<td></td>
</tr>
<tr>
<td>IXW</td>
<td>Converts a word to hexadecimal</td>
<td></td>
</tr>
<tr>
<td>IXL</td>
<td>Converts a longword to hexadecimal</td>
<td></td>
</tr>
<tr>
<td>I2B</td>
<td>Converts an unsigned decimal byte</td>
<td></td>
</tr>
<tr>
<td>I2W</td>
<td>Converts an unsigned decimal word</td>
<td></td>
</tr>
<tr>
<td>I2L</td>
<td>Converts an unsigned decimal longword</td>
<td></td>
</tr>
<tr>
<td><strong>Numeric Conversion (blank-filled)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IUB</td>
<td>Converts an unsigned decimal byte</td>
<td>Value to be converted to ASCII representation</td>
</tr>
<tr>
<td>IUB</td>
<td>Converts an unsigned decimal word</td>
<td>For byte or word conversion, FAO uses only the low-order byte or word of the longword parameter</td>
</tr>
<tr>
<td>IUL</td>
<td>Converts an unsigned decimal longword</td>
<td></td>
</tr>
<tr>
<td>ISB</td>
<td>Converts a signed decimal byte</td>
<td></td>
</tr>
<tr>
<td>ISW</td>
<td>Converts a signed decimal word</td>
<td></td>
</tr>
<tr>
<td>ISL</td>
<td>Converts a signed decimal longword</td>
<td></td>
</tr>
</tbody>
</table>

\(^1\) If a variable repeat count and/or a variable output field length is specified with a directive, parameters indicating the count and/or length must precede other parameters required by the directive.
Table 4-2 (Cont.)
Summary of FAO Directives

<table>
<thead>
<tr>
<th>Directive</th>
<th>Function</th>
<th>Parameter(s)¹</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Output String Formatting</td>
<td></td>
</tr>
<tr>
<td>!/</td>
<td>Inserts new line (CR/LF)</td>
<td>None</td>
</tr>
<tr>
<td>_</td>
<td>Inserts a tab</td>
<td></td>
</tr>
<tr>
<td>*</td>
<td>Inserts a form feed</td>
<td></td>
</tr>
<tr>
<td>!</td>
<td>Inserts an exclamation mark</td>
<td></td>
</tr>
<tr>
<td>!%S</td>
<td>Inserts the letter S if most recently</td>
<td>Address of a quadword time value to be converted to ASCII. If 0 is specified, the current system time is used.</td>
</tr>
<tr>
<td></td>
<td>converted numeric value is not l</td>
<td></td>
</tr>
<tr>
<td>!%T</td>
<td>Inserts the system time</td>
<td></td>
</tr>
<tr>
<td>!%D</td>
<td>Inserts the system date and time</td>
<td></td>
</tr>
<tr>
<td>!n&lt;</td>
<td>Defines output field width of n</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>characters. All data and directives</td>
<td></td>
</tr>
<tr>
<td></td>
<td>within delimiters are left-justified</td>
<td></td>
</tr>
<tr>
<td></td>
<td>and blank-filled within the field</td>
<td></td>
</tr>
<tr>
<td>!n&gt;c</td>
<td>Repeats the specified character in the</td>
<td></td>
</tr>
<tr>
<td></td>
<td>output string n times</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Parameter Interpretation</td>
<td></td>
</tr>
<tr>
<td>!-</td>
<td>Reuses last parameter in the list</td>
<td>None</td>
</tr>
<tr>
<td>!+</td>
<td>Skips the next parameter in the list</td>
<td></td>
</tr>
</tbody>
</table>

¹ If a variable repeat count and/or a variable output field length is specified with a directive, parameters indicating the count and/or length must precede other parameters required by the directive.
<table>
<thead>
<tr>
<th>Conversion/Substitution Type</th>
<th>Default Length of Output Field</th>
<th>Action When Explicit Output Field Length is Longer than Default</th>
<th>Action When Explicit Output Field Length is Shorter than Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hexadecimal</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Byte</td>
<td>2 (zero-filled)</td>
<td>ASCII result is right-justified and blank-filled to the specified length</td>
<td>ASCII result is truncated on the left</td>
</tr>
<tr>
<td>Word</td>
<td>4 (zero-filled)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Longword</td>
<td>8 (zero-filled)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Octal</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Byte</td>
<td>3 (zero-filled)</td>
<td>Hexadecimal or octal output is always zero-filled to the default output field length then blank-filled to specified length</td>
<td></td>
</tr>
<tr>
<td>Word</td>
<td>6 (zero-filled)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Longword</td>
<td>11 (zero-filled)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Signed or Unsigned Decimal</td>
<td>As many characters as necessary</td>
<td>ASCII result is right-justified and blank-filled to the specified length</td>
<td>Signed and unsigned decimal output fields are completely filled with asterisks(*)</td>
</tr>
<tr>
<td>Unsigned Zero-filled Decimal</td>
<td>As many characters as necessary</td>
<td>ASCII result is right-justified and zero-filled to the specified length</td>
<td></td>
</tr>
<tr>
<td>ASCII String Substitution</td>
<td>Length of input character string</td>
<td>ASCII string is left-justified and blank-filled to the specified length</td>
<td>ASCII string is truncated on the right</td>
</tr>
</tbody>
</table>
4.36.5 FAO Examples

Each of the examples on the following pages shows an FAO control string with several directives, parameters defined as input for the directives, and the calls to $FAO to format the output strings. The numbered examples illustrate:

1. $FAO macro, !AC, !AS, !AD, and !/ directives
2. $FAO macro, !!, and !AS directives, repeat count, output field length
3. $FAO macro, !UL, !XL, !SL directives
4. $FAO!L macro, !UL, !XL, !SL directives
5. $FAO!L macro, !UB, !XB, !SB directives
6. $FAO macro, !XW, !ZW, !- directives, repeat count, output field length
7. $FAO!L macro, !AS, !UB, !%S, !- directives, variable repeat count
8. $FAO macro, !n*c (repeat character), !%D directives
9. $FAO macro, !%D and !%T (with output field lengths), !n* (with variable repeat count)
10. $FAO macro, !< and !> (define field width), !AC, !AD, !UL directives

Each example is accompanied by notes, under the heading "Results". These notes show the output string created by the call to $FAO and describe in more detail some considerations for using directives. The sample output strings show delta characters (Δ) in all places where FAO output contains multiple blanks.

Each of the examples refers to the following output fields (these fields are not shown in the data areas for each example):

FAODESC:.LONG 80 ;OUTPUT BUFFER DESCRIPTOR
.LONG FAOBUF ;ADDRESS OF BUFFER
.FLABN:.BLKB 80 ;80-CHARACTER BUFFER
.FALOEN:.BLKW 1 ;RECEIVE LENGTH OF OUTPUT
.BLKW 1 ;RESERVE WORD FOR $QIO

These examples assume that each call to $FAO will be followed by a call to $QIO or to $OUTPUT to write the output string produced by FAO. The $QIO system service (and the $OUTPUT macro) require that the length be specified as a longword; therefore, an extra word is provided following the word defined to receive the length of the output string returned by $FAO.
Example 1

; CONTROL STRING AND INPUT PARAMETERS FOR EXAMPLE 1

SLEEPSTR: DESCRIPTOR <!/SAILORS: !AC !AS !AD> ; CONTROL STRING

WINKEN: .ASCIC /WINKEN/ ; COUNTED ASCII STRING
BLINKEN: DESCRIPTOR <BLINKEN> ; CHARACTER STRING DESCRIPTOR
NOD: .ASCII /NOD/ ; ASCII STRING
NODLEN: .LONG NODLEN-NOD ; LENGTH OF ASCII STRING

; CALL TO $FAO

$FAO_S CTRSTR=SLEEPSTR,OUTLEN=FAOLEN,OUTBUF=FAODESC,-
P1=#WINKEN,P2=#BLINKEN,P3=NODLEN,P4=#NOD

Results:

$FAO writes the output string into FAOBUF:

<CR><LF>SAILORS: WINKEN BLINKEN NOD

The !/ directive provides a carriage return/line feed character (shown as <CR><LF>) for terminal output.

The !AC directive requires the address of a counted ASCII string (P1 argument); the number sign (#) is required to specify the parameter, so that the PUSHL instruction used by the $FAO macro pushes the address rather than its contents.

The !AS directive requires the address of a character string descriptor (P2 argument).

The !AD directive requires two parameters: the length of the string to be substituted (P3 argument), and its address (P4 argument).

Example 2

; CONTROL STRING AND INPUT PARAMETERS FOR EXAMPLE 2

NAMESTR: DESCRIPTOR <UNABLE TO LOCATE !3(8AS)!!> ; CONTROL STRING

JONES: DESCRIPTOR <JONES> ; NAME DESCRIPTOR
HARRIS: DESCRIPTOR <HARRIS> ; NAME DESCRIPTOR
WILSON: DESCRIPTOR <WILSON> ; NAME DESCRIPTOR

; CALL TO $FAO

$FAO_S CTRSTR=NAMESTR,OUTLEN=FAOLEN,OUTBUF=FAODESC,-
P1=#JONES,P2=#HARRIS,P3=#WILSON

Results:

$FAO writes the output string into FAOBUF:

UNABLE TO LOCATE JONESHARRISWILSON!

The !3(8AS) directive contains a repeat count: 3 parameters (addresses of character string descriptors) are required. $FAO left-justifies each string into a field of 8 characters (the output field length specified).

The !! directive supplies a literal ! in the output.
SYSTEM SERVICE DESCRIPTIONS
$FAO - FORMATTED ASCII OUTPUT

If the directive were specified without an output field length, that is, if the directive had been specified as !3(AS), the 3 output fields would be concatenated, as follows:

UNABLE TO LOCATE JONESHARRISWILSON!

Examples 3, 4, and 5

; CONTROL STRINGS AND INPUT PARAMETERS FOR EXAMPLES 3, 4 AND 5

LONGSTR:
   ;CONTROL STRING (LONGWORD CONVERSION)
   DESCRIPTOR <VALUES !UL (DEC) !XL (HEX) !SL (SIGNED)>

BYTESTR:
   ;CONTROL STRING (BYTE CONVERSION)
   DESCRIPTOR <VALUES UB (DEC) XB (HEX) SB (SIGNED)>

VAL1: .LONG 200 ;DECIMAL 200
VAL2: .LONG 300 ;DECIMAL 300
VAL3: .LONG -400 ;NEGATIVE 400

; CALL TO $FAO (EXAMPLE 3)

$FAO.S CTRSTR=LONGSTR,OUTBUF=FAODESC,OUTLEN=FAOLEN,-
P1=VAL1,P2=VAL2,P3=VAL3

Results:

$FAO writes the output string:

VALUES 200 (DEC) 0000012C (HEX) -400 (SIGNED)

The longword value 200 is converted to decimal, the value 300 is converted to hexadecimal, and the value -400 is converted to signed decimal. The ASCII results of each conversion are placed in the appropriate position in the output string.

Note that the hexadecimal output string has 8 characters and is zero-filled to the left. This is the default output length for hexadecimal longwords.

; CALL TO $FAO (EXAMPLE 4)

$FAOL.S CTRSTR=LONGSTR,OUTBUF=FAODESC,OUTLEN=FAOLEN,-
PRMLST=VAL1

Results:

$FAO writes the output string:

VALUES 200 (DEC) 0000012C (HEX) -400 (SIGNED)

The results are the same as the results of example 3. However, instead of the $FAO macro, and coding each parameter on the call, the $FAOL macro points to the list of consecutive longwords, and FAO reads from the list.
$FAO writes the output string:

HEX:ΔΔΔ2710ΔΔ270F ZERO-DEC: 0010000000999

Each of the directives !2(6XW) and !2(7ZW) contain repeat counts and output lengths. First, $FAO performs the !XW directive twice, using the low-order word of the numeric parameters passed. The output length specified is 2 characters longer than the default output field width of hexadecimal word conversion, so 2 spaces are placed between the resulting ASCII strings.

The !- directive causes $FAO to back up over the parameter list. A repeat count is specified with the directive, so that $FAO skips back over two parameters; then, it uses the same two parameters for the !ZW directive. The !ZW directive causes the output string to be zero-filled to the specified length, in this example, 7 characters. Thus, there are no blanks between the output fields.
Example 7

; CONTROL STRING AND INPUT PARAMETERS FOR EXAMPLE 7

ARGSTR: DESCRIPTOR <!AS RECEIVED !UB ARG!ZS: !-!%(4UB)>

LISTA: .LONG ORION  #ADDRESS OF NAME STRING
       .LONG 3  #NUMBER OF ARGS IN LIST
       .LONG 10  #ARGUMENT 1
       .LONG 123  #ARGUMENT 2
       .LONG 210  #ARGUMENT 3

LISTB: .LONG LYRA  #ADDRESS OF NAME STRING
       .LONG 1  #NUMBER OF ARGS IN LIST
       .LONG 255  #ARGUMENT 1

ORION: DESCRIPTOR <ORION>  #PROCESS NAME

LYRA: DESCRIPTOR <LYRA>  #PROCESS NAME

; CALLS TO FAO

$FAO_S CTRSTR=ARGSTR,OUTLEN=FAOLEN,OUTBUF=FAODESC,--; PRMLST=LISTA

$FAO_S CTRSTR=ARGSTR,OUTLEN=FAOLEN,OUTBUF=FAODESC,--; PRMLST=LISTB

Results:

Following the first call to $FAO shown above, FAO writes the output string:

ORION RECEIVED 3 ARGS: 10 123 210

Following the second call, FAO writes the output string:

LYRA RECEIVED 1 ARG: 255

In each of the examples, the PRMLST argument points to a different parameter list; each list contains, in the first longword, the address of a character string descriptor. The second longword begins an argument list, with the number of arguments remaining in the list. The control string uses this second longword twice: first to output the value contained in the longword, and then to provide the repeat count to output the number of arguments in the list (the !- directive indicates that FAO should reuse the parameter).

The !%S directive provides a conditional plural. When the last value converted has a value not equal to 1, FAO outputs an "S"; if the value is a 1 (as in the second example), FAO does not output an "S".

The output field length defines a width of 4 characters for each byte value converted, to provide spacing between the output fields.
SYSTEM SERVICE DESCRIPTIONS
$FAO - FORMATTED ASCII OUTPUT

Example 8

; CONTROL STRING FOR EXAMPLE 8

TIMESTR: DESCRIPTOR `"!5*: NOW IS: !%D`'

; CALL TO $FAO

$FAO_S CTRSTR=TIMESTR,OUTLEN=FAOLEN,OUTBUF=FAODESC,-
     P1=$0

Results:

FAO writes the output string:

`>>>>> NOW IS: dd-mmm-yyyy hh:mm:ss.cc`

where dd-mmm-yyyy is the current day, month, and year, and hh:mm:ss.cc is the current time of day in hours, minutes, seconds, and hundredths of seconds.

The !5*: directive requests FAO to write five greater than (>) characters into the output string. Since there is a space after the directive, FAO also writes a space after the > characters on output.

The !%D directive requires the address of a quadword time value, which must be in the system time format. However, when the address of the time value is specified as 0, FAO uses the current date and time. For information on how to obtain system time values in the required format, see Section 3.6, "Timer and Time Conversion Services." For a detailed description of the ASCII date and time string returned, see the discussion of the Convert Binary Time to ASCII String ($ASCTIM) system service in this chapter.

Example 9

; CONTROL STRING FOR EXAMPLE 9

DAYTIMSTR: DESCRIPTOR <DATE: !11%D!*TIME: !5%T>

; CALL TO FAO

$FAO_S CTRSTR=DAYTIMSTR,OUTLEN=FAOLEN,OUTBUF=FAODESC,-
     P1=$0,P2=$5,P3=$0

Results:

FAO writes the output string:

`DATE: dd-mmm-yyyy TIME: hh:mm`

In this example, an output length of 11 bytes is specified with the !11% directive, so that FAO truncates the time from the date and time string, and outputs only the date.

The !11% directive requests that the underline character (_) be repeated the number of times specified by the next parameter. Since P2 is specified as 5, 5 underlines are written into the output string.

The !5%T directive normally returns the full system time; in this example, the !5%T directive provides an output length for the time; only the hours and minutes fields of the time string are written into the output buffer.
Example 10

; CONTROL STRING AND PARAMETERS FOR EXAMPLE 10

WIDTHSTR: DESCRIPTOR "'!25<VAR1: !AC VAL: !UL!TOTAL: !UL'

VAR1NAME: .ASCIC /INVENTORY/
VAR1: .LONG 334
VAR1TOT: .LONG 6554

VAR2NAME: .ASCIC /SALES/
VAR2: .LONG 280
VAR2TOT: .LONG 10750

; CALLS TO $FAO

$FAO_S CTRSTR=WIDTHSTR, OUTLEN=FAOLEN, OUTBUF=FAODESC,=
P1=#VAR1NAME, P2=VAR1, P3=VAR1TOT

$FAO_S CTRSTR=WIDTHSTR, OUTLEN=FAOLEN, OUTBUF=FAODESC,=
P1=#VAR2NAME, P2=VAR2, P3=VAR2TOT

Results:

Following the first call to FAO shown above, FAO writes the output string:

VAR: INVENTORY VAL: 334 TOTAL: 6554

After the second call, FAO writes the output string:

VAR: SALES VAL: 280 TOTAL: 10750

The !25< directive requests an output field width of 25 characters; the end of the field is delimited by the !> directive. Within the field defined in the example above are two directives, !AC and !UL. The strings substituted by these directives can vary in length, but the entire field always has 25 characters.

The !UL directive formats the longword passed in each example (P2 argument) and right-justifies the result in a 7-character output field.

4-89
$FORCEX

4.37 $FORCEX - FORCE EXIT

The Force Exit system service causes an Exit ($EXIT) system service call to be issued on behalf of a specified process.

Macro Format:

$FORCEX [pidadr],[prcnam],[code]

High-Level Language Format:

SYSSFORCEX([pidadr],[prcnam],[code])

pidadr
address of a longword containing the process identification of the process to be forced to exit.

prcnam
address of a character string descriptor pointing to the process name string. The name is implicitly qualified by the group number of the process issuing the force exit request.

code
longword completion code value to be used as the exit parameter. If not specified, a value of 0 is passed as the completion code.

If neither a process identification nor a process name is specified, the caller is forced to exit and control is not returned. For details on how the service interprets the PIDADR and PRCNAM arguments, see Table 3-3. Table 3-3 is in Section 3.5, "Process Control Services."

Return Status:

SS$NORMAL
Service successfully completed.

SS$ACCVIO
The process name string or string descriptor cannot be read, or the process identification cannot be written, by the caller.

SS$NONEXPR
Warning. The specified process does not exist, or an invalid process identification was specified.

SS$NOPRIV
The process does not have the privilege to force an exit for the specified process.

SS$INSMEM
Insufficient system dynamic memory is available to complete the service and the process has disabled resource wait mode with the Set Resource Wait Mode ($SETRWM) system service.
Privilege Restrictions:

User privileges are required to force an exit for:

- Other processes in the same group (GROUP privilege)
- Any other process in the system (WORLD privilege)

Resources Required/Returned:

The Force Exit system service requires system dynamic memory.

Notes:

1. The image executing in the target process follows normal exit procedures. For example, if any exit handlers have been specified, they gain control before the actual exit occurs. Use the Delete Process ($DELPrc) system service if you do not want a normal exit.

2. When a forced exit is requested for a process, a user mode AST is queued for the target process. The AST routine actually causes the Exit system service call to be issued by the target process. Because the AST mechanism is used, user mode ASTs must be enabled for the target process, or no exit occurs until ASTs are re-enabled. The process that called $FORCEx receives no notification that the exit is not being performed.

3. The $FORCEx system service completes successfully if a force exit request is already in effect for the target process but the exit is not yet completed.

For an example of the $FORCEx system service, and an explanation of the actions performed by the system when an image exits, see Section 3.5.6, "Image Exit."
$GETCHN

4.38 $GETCHN - GET I/O CHANNEL INFORMATION

The Get I/O Channel Information system service returns information about a device to which an I/O channel has been assigned. Two sets of information are optionally returned:

- The primary device characteristics
- The secondary device characteristics

In most cases the two sets of characteristic information are identical. However, the two sets provide different information in the following cases:

- If the device has an associated mailbox, the primary characteristics are those of the assigned device and the secondary characteristics are those of the associated mailbox.
- If the device is a spooled device, the primary characteristics are those of the intermediate device and the secondary characteristics are those of the spooled device.
- If the device represents a logical link on the network, the secondary characteristics contain information about the link.

Macro Format:

$GETCHN chan ,[prilen] ,[pribuf] ,[scdlen] ,[scdbuf]

High-Level Language Format:

SYSS$GETCHN(chan ,[prilen] ,[pribuf] ,[scdlen] ,[scdbuf])

chan
number of the I/O channel assigned to the device.

prilen
address of a word to receive the length of the primary characteristics.

pribuf
address of a character string descriptor pointing to the buffer that is to receive the primary device characteristics. An address of 0 (the default) implies that no buffer is specified.

scdlen
address of a word to receive the length of the secondary characteristics.

scdbuf
address of a character string descriptor pointing to buffer that is to receive the secondary device characteristics. An address of 0 (the default) implies that no buffer is specified.
SYSTEM SERVICE DESCRIPTIONS
$GETCHN - GET I/O CHANNEL INFORMATION

Return Status:

SS$_BUFFEROVF
- Service successfully completed. The device information returned
overflowed the buffer(s) provided and has been truncated.

SS$_NORMAL
- Service successfully completed.

SS$_ACVIO
- A buffer descriptor cannot be read, or a buffer or buffer length
cannot be written, by the caller.

SS$_IVCHAN
- An invalid channel number was specified, that is, a channel
number of 0 or a number larger than the number of channels
available.

SS$_NOPRIV
- The specified channel is not assigned, or was assigned from a
more privileged access mode.

Privilege Restrictions:

The Get I/O Channel Information service can be performed only on
assigned channels and from access modes that are equal to or more
privileged than the access mode from which the original channel
assignment was made.

Note:

The Get I/O Device Information ($GETDEV) system service returns
the same information as the Get I/O Channel Information system
service.

4.38.1 Format of Device Information

The $GETCHN and $GETDEV system services return information in a
user-supplied buffer. Symbolic names defined in the $DIBDEF macro
represent offsets from the beginning of the buffer. The length of the
buffer is defined in the constant DIB$K_LENGTH.

The field offset names, lengths, and contents are listed below.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Length(bytes)</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIB$L_DEVCHAR</td>
<td>4</td>
<td>Device characteristics</td>
</tr>
<tr>
<td>DIB$B_DEVCLASS</td>
<td>1</td>
<td>Device class</td>
</tr>
<tr>
<td>DIB$B_TYPE</td>
<td>1</td>
<td>Device type</td>
</tr>
<tr>
<td>DIB$W_DEVBUFSIZE</td>
<td>2</td>
<td>Device buffer size</td>
</tr>
<tr>
<td>DIB$L_DEVDEPEND</td>
<td>4</td>
<td>Device dependent information</td>
</tr>
<tr>
<td>DIB$W_UNIT</td>
<td>2</td>
<td>Unit number</td>
</tr>
<tr>
<td>DIB$W_DEVNAMOFF</td>
<td>2</td>
<td>Offset to device name string</td>
</tr>
<tr>
<td>DIB$L_PID</td>
<td>4</td>
<td>Process identification of device owner</td>
</tr>
<tr>
<td>DIB$L_OWNUIC</td>
<td>4</td>
<td>UIC of device owner</td>
</tr>
<tr>
<td>DIB$W_VPROM</td>
<td>2</td>
<td>Volume protection mask</td>
</tr>
<tr>
<td>DIB$W_ERRCNT</td>
<td>2</td>
<td>Error count</td>
</tr>
<tr>
<td>DIB$W_OPCNT</td>
<td>4</td>
<td>Operation count</td>
</tr>
<tr>
<td>DIB$W_VOLNAMOFF</td>
<td>2</td>
<td>Offset to volume label string</td>
</tr>
<tr>
<td>DIB$W_RECFSIZ</td>
<td>2</td>
<td>Blocked record size (valid for magnetic tapes when DIB$W_VOLNAMOFF is nonzero)</td>
</tr>
</tbody>
</table>
The device name string and volume label string are returned in the buffer as counted ASCII strings and must be located by using their offsets from the beginning of the buffer.

Any fields inapplicable to a particular device are returned as zeros.

For further details on the contents of this buffer, and on device-dependent information returned, see the VAX/VMS I/O User's Guide.
$GETDEV

4.39 $GETDEV - GET I/O DEVICE INFORMATION

The Get I/O Device Information system service returns information about an I/O device. This service allows a process to obtain information about a device to which the process has not assigned a channel. It returns the same information as the Get I/O Channel Information ($GETCHN) system service, as described in Section 4.38.

Macro Format:

```
$GETDEV devnam ,[prilen] ,[pribuf] ,[scdlen] ,[scdbuf]
```

High-Level Language Format:

```
SYSGETDEV(devnam ,[prilen] ,[pribuf] ,[scdlen] ,[scdbuf])
```

devnam

address of a character string descriptor pointing to the device name string. The string may be either a physical device name or a logical name. If the first character in the string is an underline character (_), the name is considered a physical device name. Otherwise, a single level of logical name translation is performed and the equivalence name, if any, is used.

prilen

address of a word to receive the length of the primary characteristics.

pribuf

address of a character string descriptor pointing to the buffer that is to receive the primary device characteristics. An address of 0 (the default) implies that no buffer is specified.

scdlen

address of a word to receive the length of the secondary characteristics.

scdbuf

address of a character string descriptor pointing to buffer that is to receive the secondary device characteristics. An address of 0 (the default) implies that no buffer is specified.

Return Status:

```
SS$ _BUFFEROVF
Service successfully completed. The device information returned overflowed the buffer(s) provided and has been truncated.
```

```
SS$ _NORMAL
Service successfully completed.
```

```
SS$ _ACCVIO
A buffer descriptor cannot be read, or a buffer or buffer length cannot be written, by the caller.
```
SYS$ _IVDEVNAM
   No device name was specified, or the device name string has invalid characters.

SYS$ _IVLOGNAM
   The device name string has a length of 0 or has more than 63 characters.

SYS$ _NONLOCAL
   Warning. The device is on a remote system.

SYS$ _NOSUCHDEV
   Warning. The specified device does not exist on the host system.
4.40 $GETJPI - GET JOB/PROCESS INFORMATION

The Get Job/Process Information system service provides accounting, status, and identification information about a specified process.

Macro Format:

$GETJPI ,pidadr ,prcnam ,itmlst,,,

High-Level Language Format:

SYS$GETJPI(/pidadr ,prcnam ,itmlst,,)

pidadr

address of a longword containing the process identification of the process for which information is requested.

prcnam

address of a character string descriptor pointing to a 1- to 15-character process name string. The process name is implicitly qualified by the group number of the process issuing the request.

itmlst

address of a list of item descriptors that describe the specific information requested and point to buffers to receive the information. The format of the list is described in Section 4.40.1. The item codes are listed in Table 4-4.

If neither a process identification nor a process name is specified, information about the calling process is returned. For details on how the service interprets the PIDADR and PRCNAM arguments, see Table 3-3. Table 3-3 is in Section 3.5, "Process Control Services."

Return Status:

SS$NORMAL

Service successfully completed.

SS$BADPARAM

The item list contains an invalid identifier; or, the caller requested information that is not in the process control block about another process.

SS$ACCVIO

The item list cannot be read, or the buffer length or buffer cannot be written, by the caller.

SS$IVLOGNAM

The process name string has a length of 0, or has more than 15 characters.

1 The first, fifth, sixth, and seventh arguments in the $GETJPI argument list are not used; they are reserved for future use.
Warning. The specified process does not exist, or an invalid process identification was specified.

The process does not have the privilege to obtain information about the specified process.

Privilege Restrictions:

User privileges are required to obtain information about:

- Other processes in the same group (GROUP privilege)
- Any other process in the system (WORLD privilege)

Note:

When a process requests information about itself, information contained in the PCB, in the process header, or in the control region of the process's virtual address space can be obtained.

When a process requests information about another process, only information contained in the PCB can be obtained.

4.40.1 Format of Item List for $GETJPI System Service

The item list used for input to the $GETJPI system service consists of one or more consecutive item descriptors. Each item descriptor in this list has the format:

```
<table>
<thead>
<tr>
<th>31</th>
<th>16</th>
<th>15</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>item code</td>
<td></td>
<td>buffer length</td>
<td></td>
</tr>
<tr>
<td></td>
<td>buffer address</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>return length address</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

**buffer length**

Length of the buffer to receive the specified information. All buffers reserved to receive information should be longwords unless otherwise indicated in Table 4-4.

**item code**

Symbolic name defining the information to be returned. The symbolic names have the format:

```
JPI$_code
```

These symbolic names are defined in the $JPIDEF macro. The codes are listed in Table 4-4.
buffer address
address of the buffer to receive the specified information. If
the buffer is too small for the requested information, $GETJPI
truncates it.

return length address
address of a word to receive the length of the information
returned. If this address is specified as 0, no length is
returned.

The list of item descriptors must be terminated by an item code of 0
or a longword of 0.

All buffers are zero-filled on return, if necessary.

For example, an item list can be coded as follows to obtain the
process identification and process name of a process:

```
GETLIST: .WORD 4
          .WORD JPI*_PID
          .LONG GETPID
          .LONG 0
          .WORD 15
          .WORD JPI*_PRCNAM
          .LONG GETPRCNAM
          .LONG PRCNAM_LEN
          .LONG 0
GETPID:  .BLKL 1
GETPRCNAM:
          .BLKB 15
PRCNAM_LEN:
          .BLKW 1
```

$LENGTH OF BUFFER
$REQUEST PID
$ADDRESS TO RECEIVE PID
$DON'T NEED LENGTH RETURN
$LENGTH OF BUFFER
$REQUEST PROCESS NAME
$ADDRESS TO RECEIVE NAME
$ADDRESS TO RECEIVE LENGTH
$END OF GETLIST
$RETURN PID HERE
$RETURN PROCESS NAME HERE
$RETURN LENGTH OF PROCESS NAME
## SYSTEM SERVICE DESCRIPTIONS

$GETJPI - GET JOB/PROCESS INFORMATION

### Table 4-4

<table>
<thead>
<tr>
<th>Item Identifier</th>
<th>Data Type</th>
<th>Location</th>
<th>Information Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>JPI$ _ACCOUNT</td>
<td>string</td>
<td>control</td>
<td>Account name string (1-8 characters)</td>
</tr>
<tr>
<td>JPI$ _APTCNT</td>
<td>value</td>
<td>PCB</td>
<td>Active page table count</td>
</tr>
<tr>
<td>JPI$ _ASTACT</td>
<td>value</td>
<td>PCB</td>
<td>Access modes with active ASTs</td>
</tr>
<tr>
<td>JPI$ _ASTCNT</td>
<td>value</td>
<td>PCB</td>
<td>Remaining AST quota</td>
</tr>
<tr>
<td>JPI$ _ASTREN</td>
<td>value</td>
<td>PCB</td>
<td>Access modes with ASTs enabled</td>
</tr>
<tr>
<td>JPI$ _ASTLM</td>
<td>value</td>
<td>PHD</td>
<td>AST limit quota</td>
</tr>
<tr>
<td>JPI$ _BIOCNT</td>
<td>value</td>
<td>PCB</td>
<td>Remaining buffered I/O quota</td>
</tr>
<tr>
<td>JPI$ _BIOLM</td>
<td>value</td>
<td>PCB</td>
<td>Buffered I/O limit quota</td>
</tr>
<tr>
<td>JPI$ _BUFIO</td>
<td>value</td>
<td>PHD</td>
<td>Count of process buffered I/O operations</td>
</tr>
<tr>
<td>JPI$ _BYTCNT</td>
<td>value</td>
<td>PCB</td>
<td>Remaining buffered I/O byte count quota</td>
</tr>
<tr>
<td>JPI$ _BYTLM</td>
<td>value</td>
<td>PCB</td>
<td>Buffered I/O byte count limit quota</td>
</tr>
<tr>
<td>JPI$ _CPULIM</td>
<td>value</td>
<td>PHD</td>
<td>Limit on process CPU time</td>
</tr>
<tr>
<td>JPI$ _CPUTIM</td>
<td>value</td>
<td>PHD</td>
<td>Accumulated CPU time (in 10-millisecond ticks)</td>
</tr>
<tr>
<td>JPI$ _CURPRIV</td>
<td>value</td>
<td>PHD</td>
<td>Quadword mask of process's current privileges</td>
</tr>
<tr>
<td>JPI$ _DPFC</td>
<td>value</td>
<td>PHD</td>
<td>Default page fault cluster size</td>
</tr>
<tr>
<td>JPI$ _DPWSCNT</td>
<td>value</td>
<td>PHD</td>
<td>Default working set size</td>
</tr>
<tr>
<td>JPI$ _DIOCNT</td>
<td>value</td>
<td>PCB</td>
<td>Remaining direct I/O quota</td>
</tr>
<tr>
<td>JPI$ _DIOLM</td>
<td>value</td>
<td>PCB</td>
<td>Direct I/O limit quota</td>
</tr>
<tr>
<td>JPI$ _DIRIO</td>
<td>value</td>
<td>PHD</td>
<td>Count of direct I/O operations for process</td>
</tr>
<tr>
<td>JPI$ _EPCS</td>
<td>value</td>
<td>PCB</td>
<td>Local event flags 0 through 31</td>
</tr>
<tr>
<td>JPI$ _EFCS</td>
<td>value</td>
<td>PCB</td>
<td>Local event flags 32 through 63</td>
</tr>
<tr>
<td>JPI$ _EFWM</td>
<td>value</td>
<td>PCB</td>
<td>Event flag wait mask</td>
</tr>
<tr>
<td>JPI$ _EXCVEC</td>
<td>address</td>
<td>control</td>
<td>Address of a list of exception vectors in the following order: primary and secondary exception vectors for kernel mode; primary and secondary exception vectors for executive mode; primary and secondary exception vectors for supervisor mode; primary and secondary exception vectors for user mode</td>
</tr>
<tr>
<td>JPI$ _FILCNT</td>
<td>value</td>
<td>PCB</td>
<td>Remaining open file quota</td>
</tr>
<tr>
<td>JPI$ _FILLM</td>
<td>value</td>
<td>PHD</td>
<td>Open file quota</td>
</tr>
<tr>
<td>JPI$ _FINALEXC</td>
<td>address</td>
<td>control</td>
<td>Address of a list of final exception vectors for kernel, executive, supervisor, then user access mode</td>
</tr>
<tr>
<td>JPI$ _FREPUVA</td>
<td>value</td>
<td>PHD</td>
<td>First free page at end of program region</td>
</tr>
<tr>
<td>JPI$ _FREPLVA</td>
<td>value</td>
<td>PHD</td>
<td>First free page at end of control region</td>
</tr>
</tbody>
</table>

1 In the Location column:

- **control** indicates that the information is in the control region of the process's virtual address space
- **PCB** indicates that the information is in the process control block
- **PHD** indicates that the information is in the process header

4-100
## SYSTEM SERVICE DESCRIPTIONS

\$GETJPI - GET JOB/PROCESS INFORMATION

### Table 4-4 (Cont.)

**Item Codes for Job/Process Information**

<table>
<thead>
<tr>
<th>Item Identifier</th>
<th>Data Type</th>
<th>Location</th>
<th>Information Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>JPIS_GPGCNT</td>
<td>value</td>
<td>PCB</td>
<td>Global page count in working set</td>
</tr>
<tr>
<td>JPIS_GRP</td>
<td>value</td>
<td>PCB</td>
<td>Group number of UIC</td>
</tr>
<tr>
<td>JPIS_LOGINTIM</td>
<td>value</td>
<td>control</td>
<td>Process execution time; returned as 64-bit system delta time value</td>
</tr>
<tr>
<td>JPIS_MEM</td>
<td>value</td>
<td>PCB</td>
<td>Member number of UIC</td>
</tr>
<tr>
<td>JPIS_OWNER</td>
<td>value</td>
<td>PCB</td>
<td>Process identification of process owner</td>
</tr>
<tr>
<td>JPIS_PAGEFLTs</td>
<td>value</td>
<td>PHD</td>
<td>Count of page faults</td>
</tr>
<tr>
<td>JPIS_PGFLQUOTA</td>
<td>value</td>
<td>PHD</td>
<td>Paging file quota</td>
</tr>
<tr>
<td>JPIS_PID</td>
<td>value</td>
<td>PCB</td>
<td>Process identification</td>
</tr>
<tr>
<td>JPIS_PPGCNT</td>
<td>value</td>
<td>PCB</td>
<td>Process page count in working set</td>
</tr>
<tr>
<td>JPIS_PRCSCNT</td>
<td>value</td>
<td>PCB</td>
<td>Count of subprocesses</td>
</tr>
<tr>
<td>JPIS_PRCLM</td>
<td>value</td>
<td>PHD</td>
<td>Subprocess quota</td>
</tr>
<tr>
<td>JPIS_PRCNAM</td>
<td>string</td>
<td>PCB</td>
<td>Process name (1-15 characters)</td>
</tr>
<tr>
<td>JPIS_PRI</td>
<td>value</td>
<td>PCB</td>
<td>Current process priority</td>
</tr>
<tr>
<td>JPIS_PRIB</td>
<td>value</td>
<td>PCB</td>
<td>Process's base priority</td>
</tr>
<tr>
<td>JPIS_PROCPRIV</td>
<td>value</td>
<td>control</td>
<td>Quadword mask of process's default privileges</td>
</tr>
<tr>
<td>JPIS_STATE</td>
<td>value</td>
<td>PDB</td>
<td>Process state</td>
</tr>
<tr>
<td>JPIS_STS</td>
<td>value</td>
<td>PCB</td>
<td>Process status</td>
</tr>
<tr>
<td>JPIS_TMBU</td>
<td>value</td>
<td>PCB</td>
<td>Termination mailbox unit number</td>
</tr>
<tr>
<td>JPIS_TQCNT</td>
<td>value</td>
<td>PCB</td>
<td>Remaining timer queue entry quota</td>
</tr>
<tr>
<td>JPIS_TQLM</td>
<td>value</td>
<td>PHD</td>
<td>Timer queue entry quota</td>
</tr>
<tr>
<td>JPIS_UIC</td>
<td>value</td>
<td>PCB</td>
<td>Process's UIC</td>
</tr>
<tr>
<td>JPIS_USERNAME</td>
<td>string</td>
<td>control</td>
<td>User name string (1-12 characters)</td>
</tr>
<tr>
<td>JPIS_VIRTPEAK</td>
<td>value</td>
<td>control</td>
<td>Peak virtual address size</td>
</tr>
<tr>
<td>JPIS_VOLUMES</td>
<td>value</td>
<td>control</td>
<td>Count of currently mounted volumes</td>
</tr>
<tr>
<td>JPIS_WSAUTH</td>
<td>value</td>
<td>PHD</td>
<td>Maximum authorized working set size</td>
</tr>
<tr>
<td>JPIS_WSPEAK</td>
<td>value</td>
<td>control</td>
<td>Working set peak</td>
</tr>
<tr>
<td>JPIS_WSQUOTA</td>
<td>value</td>
<td>PHD</td>
<td>Working set size quota</td>
</tr>
<tr>
<td>JPIS_WSSIZE</td>
<td>value</td>
<td>PHD</td>
<td>Process's current working set size</td>
</tr>
</tbody>
</table>

1 In the Location column:

- **control** indicates that the information is in the control region of the process's virtual address space
- **PCB** indicates that the information is in the process control block
- **PHD** indicates that the information is in the process header

**TYPE** \$SYS\$SHARE: JPIDEF.H

4-101
$GETMSG

4.41 $GETMSG  -  GET MESSAGE

The Get Message system service transfers a message from the system message file to the caller's buffer. This service is used by the operating system to retrieve messages based on unique message identifications and to prepare to output them.

Macro Format:

$GETMSG  msgid ,msglen ,bufadr ,[flags] ,[outadr]

High-Level Language Format:

SYSSGETMSG(msgid ,msglen ,bufadr ,[flags] ,[outadr])

msgid
identification of the message to be retrieved. Each message in the system message file has a unique identification, contained in the high-order 29 bits of system longword status codes.

msglen
address of a word to receive the length of the string returned.

bufadr
address of a character string descriptor pointing to the buffer to receive the message string. The maximum size of any message that can be returned is 256 bytes.

flags
mask defining message content. The bits in the mask and their meanings are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>Include text of message</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>Do not include text of message</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Include message identifier</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>Do not include message identifier</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>Include severity indicator</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>Do not include severity indicator</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>Include facility name</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>Do not include facility name</td>
</tr>
</tbody>
</table>

If this argument is omitted in a MACRO service call, it defaults to a value of 15, that is, all flags are set and all components of the message are returned.
outadr
address of a 4-byte array to receive the following values:

<table>
<thead>
<tr>
<th>Byte</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>Count of FAO arguments associated with message</td>
</tr>
<tr>
<td>2</td>
<td>User-specified value in message; if any</td>
</tr>
<tr>
<td>3</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

Return Status:

SS$ BUFFEROVF
Service successfully completed. The string returned overflowed the buffer provided, and has been truncated.

SS$ MSGNOTFND
Service successfully completed. The message code does not have an associated message in the file.

SS$ NORMAL
Service successfully completed.

4.41.1 Message Formats

The message identifications correspond to the symbolic names for status codes returned by system components, for example SS$ _code from system services, RMS$ _code for RMS messages, and so on.

When all bits in the FLAGS argument are set, $GETMSG returns a string in the format:

facility-severity-msgcode message-text

where:

facility identifies the component of the operating system
severity is the severity code (the low-order three bits of the status code)
msgcode is the unique message identifier
message-text is the text of the message

For example, if the MSGID argument is specified as:

MSGID=#SS$_DUPLNAM

$GETMSG returns the string:

%SYSTEM-F-DUPLNAM, duplicate process name

4-103
$GETTIM

4.42 $GETTIM - GET TIME

The Get Time system service furnishes the current system time in 64-bit format. The time is maintained in 100-nanosecond units from the system base time.

Macro Format:

$GETTIM timadr

High-Level Language Format:

SYS$GETTIM(timadr)

timadr
   address of a quadword that is to receive the current time in 64-bit format.

Return Status:

SS$NORMAL
   Service successfully completed.

SS$ACCVIO
   The quadword to receive the time cannot be written by the caller.

Note:

For an example of the $GETTIM system service, and additional details on the system time format, see Section 3.6, "Timer and Time Conversion Services."
4.43 $HIBER - HIBERNATE

The Hibernate system service allows a process to make itself inactive but to remain known to the system so that it can be interrupted, for example to receive ASTs. A hibernate request is a wait-for-wake-event request. When a wake is issued for a hibernating process with the $WAKE system service or a result of a Schedule Wakeup ($SCHDWK) system service, the process continues execution at the instruction following the Hibernate call.

Macro Format:

$HIBER_S

High-Level Language Format:

SYSSHIBER

Return Status:

SS$NORMAL
Service successfully completed.

Notes:

1. A hibernating process can be swapped out of the balance set if it is not locked into the balance set.

2. The wait state caused by this system service can be interrupted by an asynchronous system trap (AST) if (1) the access mode at which the AST is to execute is equal to or more privileged than the access mode from which the hibernate request was issued and (2) the process is enabled for ASTs at that access mode.

When the AST service routine completes execution, the system re-executes the $HIBER system service on the process's behalf. If a wakeup request has been issued for the process during the execution of the AST service routine (either by itself or another process), the process resumes execution. Otherwise, it continues to hibernate.

3. If one or more wakeup requests are issued for the process while it is not hibernating, the next hibernate call returns immediately, that is, the process does not hibernate. No count is maintained of outstanding wakeup requests.

For an example of the $HIBER system service and additional information on process hibernation, see Section 3.5.5, "Process Hibernation and Suspension." For an example of scheduled wakeup requests, see Section 3.6.6, "Scheduled Wakeups."

1 Only the "$S" macro form is provided for the Hibernate system service.
$INPUT

4.44 $INPUT - QUEUE INPUT REQUEST AND WAIT FOR EVENT FLAG

The $INPUT macro is a simplified form of the Queue I/O Request and Wait for Event Flag ($QIOW) system service. This macro queues a virtual input operation using the IOS_READVBLK function code and waits for I/O completion.

Macro Format:

$INPUT chan,length,buffer,[iosb],[efn]

chan
number of the I/O channel assigned to the device from which input is to be read.

length
length of the input buffer.

buffer
address of the input buffer.

iosb
address of a quadword I/O status block.

efn
number of the event flag to be set when the request is complete. The default is event flag 0.

Note:

The $INPUT macro has only one form. Arguments must be coded as for the $name_S macro form, but "_S" must not be included in the macro call.

Return Status, Privilege Restrictions, Resources Required/Returned, Additional Notes:

See the description of the Queue I/O Request ($QIO) system service.
4.45 $LCKPAG - LOCK PAGES IN MEMORY

The Lock Pages In Memory system service locks a page or range of pages in memory. The specified virtual pages are forced into the working set and then locked in memory. A locked page is not swapped with its working set. These pages are not candidates for page replacement and in this sense are locked in the working set as well.

Macro Format:

$\text{LCKPAG inadr, [retadr], [acmode]}

High-Level Language Format:

\text{SYS$LCKPAG(inadr, [retadr], [acmode])}

\text{inadr}
address of a 2-longword array containing the starting and ending virtual addresses of the pages to be locked. If the starting and ending virtual addresses are the same, a single page is locked. Only the virtual page number portion of the virtual addresses is used; the low-order 9 bits are ignored.

\text{retadr}
address of a 2-longword array to receive the starting and ending virtual addresses of the pages actually locked.

\text{acmode}
access mode of the locked pages. The specified access mode is maximized with the access mode of the caller. The resultant access mode must be equal to or more privileged than the access mode of the owner of each page in order to lock the page.

Return Status:

\text{SS$\_WASCLR}
Service successfully completed. All of the specified pages were previously unlocked.

\text{SS$\_WASSET}
Service successfully completed. At least one of the specified pages was previously locked in memory.

\text{SS$\_ACCVIO}

1. The input array cannot be read, or the output array cannot be written, by the caller.

2. A page in the specified range is inaccessible or does not exist.

\text{SS$\_LCKPAGFUL}
The system-defined maximum limit on the number of pages that can be locked in memory has been reached.

\text{SS$\_NOPRIV}
The process does not have the privilege to lock pages in memory.
Privilege Restrictions:

1. The user privilege PSWAPM is required to lock pages in memory.

2. The access mode of the caller must be equal to or more privileged than the access mode of the owner of the pages being locked.

Notes:

1. If more than one page is being locked, and it is necessary to determine specifically which pages had been previously locked, the pages should be locked one at a time.

2. If an error occurs while locking pages, the return array, if requested, indicates the pages that were successfully locked before the error occurred. If no pages are locked, both longwords in the return address array contain a -1.

3. Pages that are locked in memory can be unlocked with the Unlock Pages from Memory ($ULKPAG) system service. Locked pages are automatically unlocked at image exit.
4.46 $LKWSET - LOCK PAGES IN WORKING SET

The Lock Pages in Working Set system service allows a process to specify that a group of pages that are heavily used should never be replaced in the working set. The specified pages are brought into the working set if they are not already there and are locked so that they do not become candidates for replacement.

Macro Format:

$LKWSET inadr,[retadr],[acmode]

High-Level Language Format:

SYSLKWSET(inadr,[retadr],[acmode])

inadr
address of a 2-longword array containing the starting and ending virtual addresses of the pages to be locked. If the starting and ending virtual addresses are the same, a single page is locked. Only the virtual page number portion of the virtual addresses is used; the low-order 9 bits are ignored.

retadr
address of a 2-longword array to receive the starting and ending virtual addresses of the pages actually locked.

acmode
access mode of the locked pages. The specified access mode is maximized with the access mode of the caller. The resultant access mode must be equal to or more privileged than the access mode of the owner of each page in order to lock the page.

Return Status:

SS$ WASCLR
Service successfully completed. All of the specified pages were previously unlocked.

SS$ WASSET
Service successfully completed. At least one of the specified pages was previously locked in the working set.

SS$ ACCVIO
1. The input address array cannot be read, or the output address array cannot be written, by the caller.
2. A page in the specified range is inaccessible or nonexistent.

SS$ LKWSETFUL
The locked working set is full. If any more pages are locked, there will not be enough dynamic pages available to continue execution.

SS$ NOPRIV
A page in the specified range is in the system address space.
SYSTEM SERVICE DESCRIPTIONS
$LKWSET - LOCK PAGES IN WORKING SET

Privilege Restrictions:

The access mode of the caller must be equal to or more privileged than the access mode of the owner of the pages being locked.

Notes:

1. If more than one page is being locked, and it is necessary to determine specifically which pages had been previously locked, the pages should be locked one at a time.

2. If an error occurs while locking pages, the return array, if requested, indicates the pages that were successfully locked before the error occurred. If no pages are locked, both longwords in the return address array contain a -1.

3. Pages that are locked in the working set can be unlocked with the Unlock Page from Working Set ($ULWSET) system service.

For an explanation of the relationship between a process's working set and its virtual address space, see Section 3.8, "Memory Management Services."
4.47 $MGBLSC - MAP GLOBAL SECTION

The Map Global Section provides a process with access to an existing global section. Mapping a global section establishes the correspondence between pages in the process's virtual address space and the physical pages occupied by the global section.

Macro Format:

$MGBLSC inadr,[retadr],[acmode],[flags],gsdnam,[ident],
,[relpag]

High-Level Language Format:

SYS$MGBLSC(inadr,[retadr],[acmode],[flags],gsdnam,[ident],
,[relpag])

inadr

address of a 2-longword array containing the starting and ending virtual addresses in the process's virtual address space into which the section is to be mapped. The pages can be in the program (P0) region or the control (P1) region.

If the starting and ending virtual addresses are the same, a single page is mapped. Only the virtual page number portion of the virtual addresses is used; the low-order 9 bits are ignored.

retadr

address of a 2-longword array to receive the starting and ending virtual addresses of the pages into which the section was actually mapped.

acmode

access mode indicating the owner of the pages created during the mapping. The access mode is maximized with the access mode of the caller.

flags

mask defining the section type and characteristics. Flag bit settings can be ORed together to override default attributes. The flag bits for the mask are defined in the $SECDEF macro. Their meanings, and the default values they override, are:

<table>
<thead>
<tr>
<th>Flag</th>
<th>Meaning</th>
<th>Default Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEC$M WRT</td>
<td>Map section read/write</td>
<td>Map section read-only</td>
</tr>
<tr>
<td>SEC$M SYSGBL</td>
<td>System global section</td>
<td>Group global section</td>
</tr>
</tbody>
</table>

gsdnam

address of a character string descriptor pointing to the 1- to 15-character text name string for the global section. For group global sections, the global section name is implicitly qualified by the group number of the caller.
ident
address of a quadword indicating the version number of the global section and the criteria for matching the identification.

The version number is in the second longword. The version number contains two fields: a minor identification in the low-order 24 bits and a major identification in the high-order 8 bits.

The first longword specifies, in the low-order 3 bits, the matching criteria. The valid values, symbolic names by which they can be specified, and their meanings are listed below:

<table>
<thead>
<tr>
<th>Value/Name</th>
<th>Match Criteria</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 SEC$K_MATALL</td>
<td>Match all versions of the section</td>
</tr>
<tr>
<td>1 SEC$K_MATEQU</td>
<td>Match only if major and minor identifications match</td>
</tr>
<tr>
<td>2 SEC$K_MATLEQ</td>
<td>Match if the major identifications are equal and the minor identification of the mapper is less than or equal to the minor identification of the global section</td>
</tr>
</tbody>
</table>

If no address is specified, or is specified as 0 (the default), the version number and match control fields default to 0.

relpag
relative page number within the section of the first page to be mapped. If not specified or specified as 0 (the default), the global section is mapped beginning with the first virtual block in the section.

Return Status:

SS$_NORMAL
- Service successfully completed.

SS$_ACCVIO
- The input address array, the global section name or name descriptor or section identification field cannot be read, or the return address array cannot be written, by the caller.

SS$_EXQUOTA
- The process exceeded its paging file quota creating copy-on-reference pages.

SS$_INSFWSL
- The process's working set limit is not large enough to accommodate the increased virtual address space.

SS$_IVLOGNAM
- The global section name has a length of 0, or has more than 15 characters.

SS$_IVSECFLG
- A reserved flag was set.

SS$_IVSECDCTL
- The match control field of the global section identification is invalid.
SS$_NOPRIV
- The file protection mask specified when the global section was created prohibits the access or the type of access requested by the caller.

A page in the input address range is in the system address space.

SS$_NOSUCHSEC
- Warning. The specified global section does not exist.

SS$_PAGOWNVIO
- A page in the specified input address range is owned by a more privileged access mode.

SS$_VASFULL
- The process's virtual address space is full; no space is available in the page tables for the pages created to contain the mapped global section.

Privilege Restrictions:

The privilege to map a global section, and whether it may be mapped read/write or read-only, is determined by the protection mask assigned to the global section when it was created.

Resources Required/Returned:

The process's working set limit quota (WSQUOTA) must be sufficient to accommodate the increased size of the virtual address space when mapping a section. If the section pages are copy-on-reference, the process must also have sufficient paging file quota (PGFQQUOTA).

Notes:

1. When the $MGBLSC system service maps a global section, it calls the Create Virtual Address Space ($CRETVA) system service to add the pages specified by the INADR argument to the process's virtual address space.

   If the global section is of an unknown size, the process can obtain the virtual address of the first available page in the program or control region from the Get Job/Process Information ($GETJPI) system service and use the address returned as the starting address. The ending address may be a very high address (if the section is to be mapped in the program region) or a very low address (if mapped in the control region). The SCRVESC system service returns the virtual addresses of the pages created in the EETADR argument, if specified. The section is mapped from a low address to a high address, regardless of whether the section is mapped in the program or control region.

2. If an error occurs during the mapping of a global section, the return address array, if specified, indicates the pages that were successfully mapped when the error occurred. If no pages were mapped, both longwords of the return address array contain -1.

For an example of the $MGBLSC system service, and additional details on global section creation and use, see Section 3.8.6, "Sections."
$\textit{NUMTIM}$

4.48 $\textit{NUMTIM} - \textit{CONVERT} \textit{BINARY} \textit{TIME} \textit{TO} \textit{NUMERIC} \textit{TIME}$

The Convert Binary Time to Numeric Time system service converts an absolute or delta time from 64-bit system time format to binary integer date and time values. The numeric time is placed in a user-specified buffer as illustrated in Figure 4-1.

<table>
<thead>
<tr>
<th>31</th>
<th>16 15</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>month of year</td>
<td>year since 0</td>
<td></td>
</tr>
<tr>
<td>hour of day</td>
<td>day of month</td>
<td></td>
</tr>
<tr>
<td>second of minute</td>
<td>minute of hour</td>
<td></td>
</tr>
<tr>
<td></td>
<td>hundreds of second</td>
<td></td>
</tr>
</tbody>
</table>

Figure 4-1 Format of Numeric Time Buffer

Macro Format:

\[ \texttt{NUMTIM timbuf },[\texttt{timadr}] \]

High-Level Language Format:

\[ \texttt{SYS$\textit{NUMTIM}(timbuf},[\texttt{timadr}] \) \]

timbuf

address of a 7-word buffer to receive the date and time information.

timadr

address of a 64-bit time value to be converted. If not specified, or specified as 0, the current system time is used. A positive time value represents an absolute time. A negative time value indicates a delta time.

Return Status:

\texttt{SS$\_\texttt{NORMAL}}

- Service successfully completed.

\texttt{SS$\_\texttt{ACCVIO}}

- The 64-bit time value cannot be read, or the numeric buffer specified cannot be written, by the caller.

\texttt{SS$\_\texttt{IVTIME}}

- The specified delta time is equal to or greater than 10,000 days.
Note:

If a delta time is specified, the year and month fields of the information returned are zero. The day field contains the integer number of days specified by the delta time; it must be less than 10,000 days.
$OUTPUT

4.49  $OUTPUT - QUEUE OUTPUT REQUEST AND WAIT FOR EVENT FLAG

The $OUTPUT macro is a simplified form of the Queue I/O Request and Wait for Event Flag ($SQIOW) system service. This macro performs a virtual output operation using the IOS_WRITEVBLK function code and waits for I/O completion.

Macro Format:

$OUTPUT chan ,length ,buffer ,[iosb] ,[efn]

chan
number of the I/O channel assigned to the device to which output is to be written.

length
length of the output buffer.

buffer
address of the output buffer.

iosb
address of quadword I/O status block.

efn
number of the event flag to be set when the request is complete. The default is event flag 0.

Note:

The $OUTPUT macro has only one form. Arguments must be coded as for the $name_S macro form, but "_S" must not be included in the macro call.

Return Status, Privilege Restrictions, Resources Required/Returned,
Additional Notes:

See the description of the Queue I/O Request ($SQIO) system service for details.
4.50  **$PURGWS** — PURGE WORKING SET

The Purge Working Set system service enables a process to remove pages from its current working set to reduce the amount of physical memory occupied by the current image.

**Macro Format:**

```cobol
$PURGWS inadr
```

**High-Level Language Format:**

```cobol
SYS$PURGWS(inadr)
```

**inadr**

address of a 2-longword array containing the starting and ending virtual addresses of the pages to be potentially purged from the working set. The $PURGWS system service locates pages within this range that are in the current working set and removes them.

If the starting and ending virtual addresses are the same, only that single page is a candidate for purging. Only the virtual page number portion of the virtual addresses is used; the low-order 9 bits are ignored.

**Return Status:**

**SYS$_NORMAL**

- Service successfully completed.

**SYS$_ACCVIO**

- The input address array cannot be read by the caller.

**Note:**

To purge the entire working set, the caller can specify a range of pages from 0 through 7FPPPFPFP. The image continues executing, and pages that are needed are brought back into the working set as the page faults occur.
$PUTMSG

4.51 $PUTMSG - PUT MESSAGE

The Put Message system service is a generalized message formatting and
output routine used by the operating system to write informational and
error messages to user processes.

Macro Format:

    $PUTMSG msgvec , [actrtn] , [facnam]

High-Level Language Format:

    SYS$PUTMSG(msgvec , [actrtn] , [facnam])

msgvec
    address of a message argument vector that lists the message
    identifications of messages to be output and FAO arguments
    associated with each message, if any. The format of the message
    vector is described in Section 4.51.1, below.

actrtn
    address of the entry mask of a user-specified action routine to
    receive control during message processing. The action routine
    receives control after a message is formatted but before it is
    actually written to the user. If no address is specified, or
    specified as 0 (the default), it indicates that there is no
    action routine.

facnam
    address of a character string descriptor pointing to the facility
    name to be used in the first or only message formatted by
    $PUTMSG.
    
    If not specified, the default facility name associated with the
    message is used in the first message.

Return Status:

    SS$NORMAL
    Service successfully completed.

Note:

    The $PUTMSG system service disables AST delivery while it is
    executing to prevent recursive entry.
4.51.1 Format of the Message Argument Vector

The general format of a message argument vector is as shown below. Messages with facility codes of either 0 (system status codes) or 1 (RMS status codes) vary from the basic format.

<table>
<thead>
<tr>
<th>31</th>
<th>16 15</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>message flags</td>
<td>argument count</td>
<td>first message identification</td>
</tr>
<tr>
<td>message flags</td>
<td>FAO count</td>
<td>FAO arguments</td>
</tr>
<tr>
<td>next message identification</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**argument count**
specifies the total number of longwords in the message vector.

**message flags**
specifies a mask defining the portions of the message(s) to be requested from the $GETMSG system service. If not specified, $PUTMSG calls $GETMSG requesting that all fields in the message text be returned. If a mask is specified, it is passed to $GETMSG as the FLAGS argument. The bits in the mask and their meanings are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>Include text of message</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>Do not include text of message</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Include message identifier</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>Do not include message identifier</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>Include severity level indicator</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>Do not include severity level indicator</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>Include facility name</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>Do not include facility name</td>
</tr>
</tbody>
</table>

Bits 4 through 15 must be zeros.

**first message identification**
32-bit numeric value that uniquely identifies the first, or only, message. Messages can be identified by symbolic names defined for system return status codes, RMS status codes, and so on.

**FAO count**
number of FAO arguments, if any, that follow in the message vector. The FAO argument count is required for all message identifiers for which the facility code is other than 0 (the system) or 1 (RMS). If a message with any other facility code has no associated FAO arguments, the FAO argument count must be specified as 0, unless the message identifier is the final item in the message vector.
message flags
new mask for the $GETMSG flags, defining a new default for all
subsequent messages.

FAO arguments...
FAO arguments required by the message.

next message identification...
identification of next associated message, if messages are linked
in a series. $PUTMSG returns the first message with the percent
sign (%) prefix in front of the message. By convention, messages
after the first message in a series are prefixed with a hyphen
(-).

Message identifications for system status codes, system exception
condition values, and RMS status codes are handled as follows:

1. If the status code is a system message (that is, it has a
facility code of 0), neither an FAO argument count nor FAO
arguments can be specified. Each longword in the list
(following the first message identification) is treated as an
additional message identification.

2. If the message identification is a system exception message
number (for example, SS$_COMPAT), the FAO arguments for the
message must immediately follow the message identification in
the message vector. $PUTMSG determines the count of FAO
arguments from the message number.

Note that the format of the message argument vector for an
exception condition status code is identical to the signal
array argument list passed to a condition handler when the
system signals an exception condition.

3. If the message identification is an RMS status code (that is,
it has a facility code of 1), you must specify a second
longword following the status code in place of the FAO
argument count. This longword is reserved for an RMS status
value (STV) for those RMS messages that have status values
associated with them. If the status code has no STV value
associated with it, $PUTMSG ignores the second longword.
$PUTMSG uses the STV value as an FAO argument or as another
message identification, depending on the value of the RMS
message number.

No FAO arguments can be specified for RMS status codes. If
specified, $PUTMSG treats them as additional message
identifiers.

The following example shows a message argument vector that requests
$PUTMSG to output:

1. The complete message associated with the system status code
SS$ _ABORT

2. The complete message associated with the system status code
RMS$ _FNF

VECTOR: .LONG 3 $ARGUMENT COUNT
          .LONG SS$ _ABORT $ABORT MESSAGE
          .LONG RMS$ _FNF $FILE NOT FOUND MESSAGE
          .LONG 0 $IGNORED
  $PUTMSG_S MSGVEC=VECTOR

4-120
SYSTEM SERVICE DESCRIPTIONS

$PUTMSG - PUT MESSAGE

When this message vector has been processed, the following messages are written to the current SYS$OUTPUT device (and to SYS$ERROR, if it is different):

%SYSTEM-F-ABORT, abort
-RMS-E-FNF, file not found

4.51.2 Using the $PUTMSG System Service

$PUTMSG retrieves a message from the system message file by calling the Get Message ($GETMSG) system service and formats the message by calling the Formatted ASCII Output ($FAO) system service, if necessary. If the caller specifies an action routine to receive control, the action routine is called before $PUTMSG writes each formatted message to the process's current output device. If the process's error device is different than the output device, $PUTMSG writes the message to the error device as well.

4.51.2.1 $GETMSG Processing - The $GETMSG system service returns a message string based on the numeric status code value passed to it. The content of the string returned depends on the flags, if any, specified in the message argument vector. You can request that the message include or not include the facility name, severity level, message code, or text. The following example shows a message vector that requests only the text portion of the message associated with the system status code SS$DUPLNAM:

VECTOR: .WORD 1
.WORD ^B0001
.LONG SS$DUPLNAM

If this message vector is specified for a call to $PUTMSG, $PUTMSG outputs the message:

duplicate process name

$GETMSG uses the facility code in the message identification to obtain the facility name string to insert in a message. Each system component has a unique code. The facility code is contained in bits 16 through 27 of the message identification. For example, the system has facility code of 0, the command interpreter is 1, the debugger is 2, and so on.

You can override the facility name by specifying the FACNAM argument to $PUTMSG. For example:

FAC: DESCRIPTOR <HELLO>
VECTOR: .LONG 1
.LONG SS$NOPRIV

$PUTMSG_S MSGVEC=VECTOR;FACNAM=FAC

This call to $PUTMSG results in the message:

%HELLO-F-NOPRIV, no privilege for attempted operation
SYSTEM SERVICE DESCRIPTIONS
$PUTMSG - PUT MESSAGE

You can modify a facility code in a message identification before calling $PUTMSG by changing bits 16 through 27. For example, a system status code can be specified as follows:

.long 2016!SS$_.code

In this example, the facility number 2 is inserted in the message identification. You can override the facility name string DEBUG in the message by specifying message flags in the argument vector to suppress the facility name, or you can use the FACNAM argument to $PUTMSG to specify an alternate facility name.

This technique allows you to use shared system message codes that have associated FAO arguments. If you do not modify the facility number in the message identifications, you cannot specify FAO arguments.

When a message identification contains an unknown facility code, $GETMSG places the string NONAME in place of the facility name in the message string.

4.51.2.2 $FAO Processing - If the string returned by $GETMSG contains any FAO directives, and if the facility code is other than 0 or 1, $PUTMSG calls the $FAO system service to format the message. $PUTMSG calls $FAO with the argument count and arguments specified in the message argument vector.

The FAO argument count, if any, for a message is indicated in the message file that defines the message text. The message text itself contains embedded FAO directives. You can examine the message text to determine the arguments required by FAO. For example, the message text associated with the system status code SHR$$_BEGIN is defined as:

!AS beginning

This text requires the address of a character string descriptor pointing to the text to be substituted in place of the FAO directive !AS. (For details on how to use FAO, and how to specify arguments for other FAO directives, see the description of the $FAO system service.)

To use $PUTMSG to access and/or output a system shared message that has FAO arguments associated with it, you must change the facility code. The following example shows a message vector, including the FAO argument count and argument, to output the message associated with the status code SHR$$_BEGIN.

VECTOR: .word 3
.word 'BOO001
.long 2016!SHR$$_BEGIN
.long 1
.long NAME
.long NAME

NAME: DESCRIPTOR <PUTMSG tests>

When $PUTMSG is called with this message vector, it displays the line:

PUTMSG tests beginninns

Note that the facility code in the message identification is modified to allow the specification of FAO arguments; and that the message flags in the second word of the vector suppresses the printing of facility name, severity level, and message code.
4.51.2.3 The Action Routine - The action routine, if any, is called as a normal procedure each time a message is formatted, but before it is actually output. The action routine receives as an argument the address of a character string descriptor pointing to the formatted message. The action routine can access the message text, scan it, write it to a user-specified file or device, modify it, and so on.

On return from the action routine, $PUTMSG examines the completion code from the routine specified in Register 0. If the completion code indicates success (any odd numeric value), $PUTMSG outputs the message to the current output and error devices. If the completion code indicates non-success (any even numeric value), $PUTMSG does not output the message.
$QIO

4.52 $QIO - QUEUE I/O REQUEST

The Queue I/O Request system service initiates an input or output operation by queueing a request to a channel associated with a specific device. Control returns immediately to the issuing process which can synchronize I/O completion in one of three ways:

1. Specify the address of an AST routine that is to execute when the I/O completes.
2. Wait for a specified event flag to be set.
3. Poll the specified I/O status block for a completion status.

The event flag and I/O status block, if specified, are cleared before the I/O request is queued.

Macro Format:

$QIO [efn], chan, func, [iosb], [astadr], [astprm], [p1], [p2], [p3], [p4], [p5], [p6]

High-Level Language Format:

SYSSQIO([efn], chan, func, [iosb], [astadr], [astprm], [p1], [p2], [p3], [p4], [p5], [p6])

efn
number of the event flag that is to be set at request completion. If not specified, it defaults to 0.

chan
number of the I/O channel assigned to the device to which the request is directed.

func
function code and modifier bits that specify the operation to be performed. The code is expressed symbolically. For reference purposes, the function codes are listed in Appendix A. Complete details on valid I/O function codes and parameters required by each are documented in the VAX/VMS I/O User's Guide.

iosb
address of a quadword I/O status block that is to receive final completion status.

astadr
address of the entry mask of an AST service routine to be executed when the I/O completes. If specified, the AST routine executes at the access mode from which the $QIO service was requested.

astprm
AST parameter to be passed to the AST service routine.
SYSTEM SERVICE DESCRIPTIONS

$QIO - QUEUE I/O REQUEST

pl to p6
optional device- and function-specific I/O request parameters.

The first parameter may be specified as Pl or PlV, depending on
whether the function code requires an address or a value,
respectively. If the keyword is not used, Pl is the default,
that is, the argument is considered an address.

P2 through Pn are always interpreted as values.

Return Status:

SS$ NORMAL
- Service successfully completed. The I/O request packet was
  successfully queued.

SS$ ABORT
- A network logical link was broken.

SS$ ACCVIO
- The I/O status block cannot be written by the caller.

This status code may also be returned if parameters for
device-dependent function codes are incorrectly specified.

SS$ EXQUOTA
- The process has exceeded its buffered I/O quota, direct I/O
  quota, or buffered I/O byte count quota and has disabled resource
  wait mode with the Set Resource Wait Mode ($SETRWM) system
  service. Or, the process has exceeded its AST limit quota.

SS$ ILLEPC
- An illegal event flag number was specified.

SS$ INSFMEM
- Insufficient system dynamic memory is available to complete the
  service and the process has disabled resource wait mode with the
  Set Resource Wait Mode ($SETRWM) system service.

SS$ ICHAN
- An invalid channel number was specified, that is, a channel
  number of 0 or a number larger than the number of channels
  available.

SS$ NOPRIV
- The specified channel does not exist, or was assigned from a more
  privileged access mode.

SS$ UNASEPC
- The process is not associated with the cluster containing the
  specified event flag.

Privilege Restrictions:

The Queue I/O Request system service can be performed only on
assigned I/O channels and only from access modes that are equal
to or more privileged than the access mode from which the
original channel assignment was made.

See I/O User Guide 2-14

4-125
SYSTEM SERVICE DESCRIPTIONS
$QIO - QUEUE I/O REQUEST

Resources Required/Returned:

1. Queued I/O requests use the process's quota for buffered I/O (BIOLM) or direct I/O (DIOLM); the process's buffered I/O byte count (BYTLM) quota; and, if an AST service routine is specified, the process's AST limit quota (ASTLM).

2. System dynamic memory is required to construct a data base to queue the I/O request. Additional memory may be required on a device-dependent basis.

Notes:

1. The specified event flag is set if the service terminates without queuing an I/O request.

2. The I/O status block has the format:

```
+--------------+   +--------------+   +--------------+
| count        |   | status        |
| device-dependent information   |
| status        |   | byte count    |
| is the completion status of the I/O request. |
| byte count    |   | is the number of bytes actually transferred. |
| device and function dependent information |
| varies according to the device and operation being performed. The information returned for each device and function code is documented in the VAX/VMS I/O User's Guide. |
```

3. Many services return character string data and write the length of the data returned in a word provided by the caller. Function codes for the $QIO system service (and the LENGTH argument of the $OUTPUT system service) require length specifications in longwords. If lengths returned by other services are to be used as input parameters for $QIO requests, a longword should be reserved to ensure that no error occurs when $QIO reads the length.

4. For information on performing input and output operations on a network, see the DECnet-VAX User's Guide.

For examples of the $QIO system service, including the use of event flags, AST service routines, and an I/O status block, see Section 3.4, "Input/Output Services."
4.53 $QIOW - QUEUE I/O REQUEST AND WAIT FOR EVENT FLAG

The Queue I/O Request and Wait for Event Flag system service combines the $QIO and $WAITFR (Wait for Single Event Flag) system services. It can be used when a program must wait for I/O completion.

Macro Format:

$QIOW [efn],chan,func,[iosb],[astadr],[astprm]
   ,[p1],[p2],[p3],[p4],[p5],[p6]

High-Level Language Format:

SYS$QIOW([efn],chan,func,[iosb],[astadr],[astprm]
   ,[p1],[p2],[p3],[p4],[p5],[p6])


efn
number of the event flag that is to be set at request completion. If not specified, it defaults to 0.

chan
number of the I/O channel assigned to the device to which the request is directed.

func
function code and modifier bits that specify the operation to be performed. The code is expressed symbolically.

iosb
address of a quadword I/O status block that is to receive final completion status.

astadr
address of the entry mask of an AST service routine to be executed when the I/O completes. If specified, the AST routine executes at the access mode from which the $QIO service was requested.

astprm
AST parameter to be passed to the AST completion routine.

pl to p6
optional device- and function-specific I/O request parameters.

Return Status, Privilege Restrictions, Resources Required/Returned, Notes:

See the description of the $QIO system service for details.

4-127
$READEF

4.54 $READEF - READ EVENT FLAGS

The Read Event Flags system service returns the current status of all 32 event flags in a local or common event flag cluster.

Macro Format:

$READEF efn ,state

High-Level Language Format:

SYS$READEF(efn ,state)

efn
number of any event flag within the cluster to be read. A flag number of 0 through 31 specifies cluster 0, 32 through 63 specifies cluster 1, and so forth.

state
address of a longword to receive the current status of all event flags in the cluster.

Return Status:

SS$_WASCLR
Service successfully completed. The specified event flag is clear.

SS$_WASSET
Service successfully completed. The specified event flag is set.

SS$_ACCVIO
The longword that is to receive the current state of all event flags in the cluster cannot be written by the caller.

SS$_ILLEFC
An illegal event flag number was specified.

SS$_UNASEFC
The process is not associated with the cluster containing the specified event flag.
4.55 $RESUME - RESUME PROCESS

The Resume Process system service causes a process previously suspended by the Suspend Process ($SUSPND) system service to resume execution, or cancels the effect of a subsequent suspend request.

**Macro Format:**

$RESUME [pidadr] ,[prcnam]

**High-Level Language Format:**

SYS$RESUME([pidadr] ,[prcnam])

**pidadr**
address of a longword containing the process identification of the process to be resumed.

**prcnam**
address of a character string descriptor pointing to the 1- to 15-character process name string. The process name is implicitly qualified by the group number of the process issuing the resume request.

If neither a process identification nor a process name is specified, the resume request is for the caller. For details on how the service interprets the PIDADR and PRCNAM arguments, see Table 3-3. Table 3-3 is in Section 3.5, "Process Control Services."

**Return Status:**

**SS$_NORMAL**
Service successfully completed.

**SS$_ACCVIO**
The process name string or string descriptor cannot be read, or the process identification cannot be written, by the caller.

**SS$_IVLOGNAM**
The specified process name has a length of 0, or has more than 15 characters.

**SS$_NONEXPR**
Warning. The specified process does not exist, or an invalid process identification was specified.

**SS$_NOPRIV**
The process does not have the privilege to resume the execution of the specified process.
Privilege Restrictions:

User privileges are required to resume execution of:

- Other processes in the same group (GROUP privilege)
- Any other process in the system (WORLD privilege)

Note:

If one or more resume requests are issued for a process that is not suspended, a subsequent suspend request completes immediately, that is, the process is not suspended. No count is maintained of outstanding resume requests.

For more information on process suspension see Section 3.5.5, "Process Hibernation and Suspension."
4.56  $SCHDWK - SCHEDULE WAKEUP

The Schedule Wakeup system service schedules the awakening of a process that has placed itself in a state of hibernation with the Hibernate ($SHIBER) system service. A wakeup can be scheduled for a specified absolute time or for a delta time. Optionally, the request can specify that the wakeup is to be repeated at fixed intervals.

Macro Format:

$SCHDWK [pidadr],[prcnam],daytim,[reptim]

High-Level Language Format:

SYS$SCHDWK([pidadr],[prcnam],daytim,[reptim])

pidadr
address of a longword containing the process identification of the process to be awakened.

prcnam
address of a character string descriptor pointing to the 1- to 15-character process name string. The process name is implicitly qualified by the group number of the process issuing the schedule wakeup request.

daytim
address of a quadword containing the expiration time in the system 64-bit time format. A positive time value indicates an absolute time at which the specified process is to be awakened. A negative time value indicates an offset (delta time) from the current time.

reptim
address of a quadword containing the time interval (expressed in delta time format) at which to repeat the wakeup request. If not specified, it defaults to 0, which indicates that the request is not to be repeated.

If neither a process identification nor a process name is specified, the scheduled wakeup request is for the caller. For details on how the service interprets the PIDADR and PRCNAM arguments, see Table 3-3. Table 3-3 is in Section 3.5, "Process Control Services."

Return Status:

S9$NORMAL
Service successfully completed.

S9$ACCVO
The expiration time, repeat time, process name string or string descriptor cannot be read; or the process identification cannot be written, by the caller.

S9$EXQUOTA
The process has exceeded its AST limit quota.
SYSTEM SERVICE DESCRIPTIONS
$SCHDWK - SCHEDULE WAKEUP

$S$INSFNEM
Insufficient system dynamic memory is available to allocate a
timer queue entry and the process has disabled resource wait mode
with the Set Resource Wait Mode ($SETRWM) system service.

$S$IVLOGNAM
The process name string has a length of 0 or has more than 15
characters.

$S$IVTIME
The specified delta repeat time was a positive value, or was
equal to or greater than 10,000 days. Or, an absolute expiration
time or absolute time plus delta repeat time is less than the
current time.

$S$NONEXPR
Warning. The specified process does not exist, or an invalid
process identification was specified.

$S$NOPRIV
The process does not have the privilege to schedule a wakeup
request for the specified process.

Privilege Restrictions:

User privileges are required to schedule wakeup requests for:

- Other processes in the same group (GROUP privilege)
- Any other process in the system (WORLD privilege)

Resources Required/Returned:

A scheduled wakeup request uses the caller’s AST limit quota
(ASTLM) and requires system dynamic memory to allocate a timer
queue entry.

Notes:

1. If one or more scheduled wakeup requests are issued for a
   process that is not hibernating, a subsequent hibernate
   request by the target process completes immediately, that is,
   the process does not hibernate. No count is maintained of
   outstanding wakeup requests.

2. Scheduled wakeup requests that have not yet been processed
   can be canceled with the Cancel Wakeup ($SCANWAK) system
   service.

For an example of the $SCHDWK system service, and for information on
how to format a system time value for input to this service, see
Section 3.6, "Timer and Time Conversion Services." For more
information on process hibernation and waking, see Section 3.5,
"Process Control Services."
4.57 $SETAST - SET AST ENABLE

The Set AST Enable system service enables or disables the delivery of ASTs for the access mode from which the service call was issued.

Macro Format:

$SETAST enbflg

High-Level Language Format:

SYS$SETAST(enbflg)

enbflg

AST enable indicator. A value of 1 enables AST delivery for the calling access mode. A value of 0 disables AST delivery.

Return Status:

SS$WASCLR

Service successfully completed. AST delivery was previously disabled for the calling access mode.

SS$WASSET

Service successfully completed. AST delivery was previously enabled for the calling access mode.

Notes:

1. When an image is executing in user mode, the system keeps ASTs enabled for all higher access modes. If a higher access mode disables AST delivery, it should reenable ASTs for its own access mode before returning to a lower access mode.

2. If an AST is queued for an access mode that has disabled AST delivery, the system cannot deliver ASTs to less privileged access modes until the access mode reenables AST delivery.

For additional notes on AST delivery and the usage of ASTs, see Section 3.2, "Asynchronous System Trap (AST) Services."
$SETEF

4.58 $SETEF - SET EVENT FLAG

The Set Event Flag system service sets an event flag in a local or common event flag cluster to 1. Any processes waiting for the event flag are made runnable.

Macro Format:

$SETEF efn

High-Level Language Format:

SYS$SETEF(efn)

efn   number of the event flag to be set.

Return Status:

SS$_WASCLR
   Service successfully completed. The specified event flag was previously 0.

SS$_WASSET
   Service successfully completed. The specified event flag was previously 1.

SS$_ILLEFC
   An illegal event flag number was specified.

SS$_UNASEFC
   The process is not associated with the cluster containing the specified event flag.

For an example of the $SETEF system service and more information on event flags and event flag clusters, see Section 3.1, "Event Flag Services."
4.59 \$SETEXV - SET EXCEPTION VECTOR

The Set Exception Vector system service assigns a condition handler address to an exception vector or cancels an address previously assigned to a vector.

Macro Format:

\$SETEXV [vector] ,[address] ,[acmode] ,[prvhn]

High-Level Language Format:

SYS\$SETEXV([vector] ,[address] ,[acmode] ,[prvhn])

vector

vector number. A value of 0 (the default) indicates that the primary vector is to be modified. A value of 1 indicates the secondary vector is to be modified. A value of 2 indicates that a last chance exception vector is to be modified.

address

condition handler address. If not specified, or specified as 0, it indicates that there is no condition handler or that the vector is to be canceled. If an address is specified, it is the address of the entry mask of the condition handler.

acmode

access mode for which the exception vector is to be modified. The access mode of the caller is maximized with the specified access mode to determine which vector to modify.

prvhn

address of a longword to receive the previous contents of the vector.

Return Status:

\$S\$ NORMAL
Service successfully completed.

\$S\$ ACCVIO
The longword that is to receive the previous contents of the vector cannot be written by the caller.

Privilege Restrictions:

A process cannot modify a vector associated with a more privileged access mode.
Notes:

1. Exception handlers are normally declared on the procedure call stack.

2. The primary exception vector and the last chance exception vector are used by the system debugger. The command interpreter uses the last chance exception vector.

3. User mode exception vectors are canceled at image exit.

Condition handling, and conventions for coding condition handling routines, are described in Section 3.7, "Condition Handling Services."
4.60 $SETIMR - SET TIMER

The Set Timer system service allows a process to schedule the setting of an event flag and/or the queuing of an AST at some future time. The time for the event can be specified as an absolute time or as a delta time.

Macro Format:

$SETIMR [efn] ,daytim ,[astadr] ,[reqidt]

High-Level Language Format:

SYS$SETIMR([efn] ,daytim ,[astadr] ,[reqidt])

efn

event flag number of the event flag to set when the time interval expires. If not specified, it defaults to 0.

daytim

address of the quadword expiration time. A positive time value indicates an absolute time at which the timer is to expire. A negative time value indicates an offset (delta time) from the current time.

astadr

address of the entry mask of an AST service routine to be called when the time interval expires. If not specified, it defaults to 0, indicating no AST is to be queued.

reqidt

number indicating a request identification. If not specified, it defaults to 0. A unique request identification can be specified in each set timer request; or the same identification can be given to related timer requests. The identification can be used later to cancel the timer request(s). If an AST service routine is specified, the identification is passed as the AST parameter.

Return Status:

SS$_NORMAL
Service successfully completed.

SS$_ACCVIO
The expiration time cannot be read by the caller.

SS$_EXQUOTA
The process exceeded its quota for timer entries or its AST limit quota. Or, there is insufficient system dynamic memory to complete the request and the process has disabled resource wait mode with the Set Resource Wait Mode ($SETRMW) system service.

SS$_ILLEFC
An illegal event flag number was specified.
SYSTEM SERVICE DESCRIPTIONS

$SETIMR - SET TIMER

$SS_INSFMEM
Insufficient dynamic memory is available to allocate a timer queue entry and the process has disabled resource wait mode with the Set Resource Wait Mode ($SETRWM) system service.

$SS_INVTIME
The specified absolute expiration time has already passed, or was specified as 0.

$SS_UNASEFC
The process is not associated with the cluster containing the specified event flag.

Resources Required/Returned:

1. The Set Timer system service requires dynamic memory.

2. The Set Timer system service uses the process's quota for timer queue entries (TQELM) and, if an AST service routine is specified, the process's AST limit quota (ASTLM).

Notes:

1. The access mode of the caller is the access mode of the request and of the AST.

2. The Convert ASCII String to Binary Time ($BINTIM) system service converts a specified ASCII string to the quadword time format required as input to the $SETIMR service.

For examples of the $SETIMR system service, see Section 3.6, "Timer and Time Conversion Services." For an example of an AST service routine, see Section 3.2, "AST (Asynchronous System Trap) Services."
4.61 $SETPRA - SET POWER RECOVERY AST

The Set Power Recovery AST system service establishes a routine to receive control using the AST mechanism after a power recovery is detected.

Macro Format:

$SETPRA astadr , [acmode]

High-Level Language Format:

SYSS$SETPRA(astadr , [acmode])

astadr
address of the entry mask for a power recovery AST routine. An address of 0 indicates that power recovery AST notification for the process is disabled.

acmode
access mode at which the power recovery AST routine is to execute. The specified access mode is maximized with the access mode of the caller to determine the access mode to use.

Return Status:

SS$NORMAL
Service successfully completed.

SS$EXQUOTA
The process exceeded its quota for outstanding AST requests.

Resources Required/Returned:

The $SETPRA system service uses the process's AST limit quota (ASTLM).

Notes:

1. The AST parameter contains the amount of time that the power was off, in hundredths of seconds.

2. Only one power recovery AST routine can be specified for a process. The AST entry point address is cleared at image exit. A power recovery AST routine is executed only once; it must specifically re-establish itself to receive control for multiple power recovery conditions.

3. The entry and exit conventions for the power recovery AST routine are the same as for all AST service routines. These conventions are described in Section 3.2, "Asynchronous System Trap (AST) Services."
$SETPRI

4.62 $SETPRI - SET PRIORITY

The Set Priority system service changes a process's base priority. The system scheduler uses the base priority to determine the order in which executable processes are to run.

Macro Format:

$SETPRI [pidadr] ,[prcnam] ,pri ,[prvpri]

High-Level Language Format:

SYSSSETPRI([pidadr] ,[prcnam] ,pri ,[prvpri])

pidadr
   address of the process identification of the process whose priority is to be set.

prcnam
   address of a character string descriptor pointing to a 1- to 15-character process name string. The process name is implicitly qualified by the group number of the process issuing the set priority request.

pri
   new base priority to be established for the process. The new priority is contained in bits 0 through 4 of the argument.

Normal priorities are in the range 0 through 15, and time-critical priorities are in the range 16 through 31.

If the specified priority is higher than the caller's priority, and if the caller does not have the privilege to set the target process's priority to a value higher than its own, the caller's priority is used.

prvpri
   address of a longword to receive the previous base priority of the specified process.

If neither a process identification nor a process name is specified, the set priority request is for the caller. For details on how the service interprets the PIDADR and PRCNAM arguments, see Table 3-3. Table 3-3 is in Section 3.5, "Process Control Services."

Return Status:

SS$_NORMAL
   Service successfully completed.

SS$_ACCVIO
   The process name string or string descriptor cannot be read, or the process identification or previous priority longword cannot be written, by the caller.
SS$_IVLOGNAM
The process name string has a length of 0, or has more than 15 characters.

SS$_NONEXPR
Warning. The specified process does not exist, or an invalid process identification was specified.

SS$_NOPRIV
The process does not have the privilege to set the specified priority for the specified process.

Privilege Restrictions:

User privileges are required to:

- Change the priority for other processes in the same group (GROUP privilege)
- Change the priority for any other process in the system (WORLD privilege)
- Set any process's priority to a value greater than one's own initial base priority (ALTPRI privilege)

Note:

A process's base priority remains in effect until specifically changed or until the process is deleted.
4.63  $SETPRN - SET PROCESS NAME

The Set Process Name system service allows a process to establish or to change its own process name.

**Macro Format:**

```
$SETPRN [prcnam]
```

**High-Level Language Format:**

```
SYS$SETPRN([prcnam])
```

**prcnam**

address of a character string descriptor pointing to the 1- to 15-character process name string. The process name is implicitly qualified by the group number of the caller. If not specified, or specified as 0, the process's current name is deleted.

**Return Status:**

- **SS$NORMAL**
  - Service successfully completed.

- **SS$ACCVIO**
  - The process name string or string descriptor cannot be read by the caller.

- **SS$DUPLNAM**
  - The specified process name duplicates one already specified within that group.

- **SS$IVLOGNAM**
  - The specified process name has a length of 0 or has more than 15 characters.

**Notes:**

1. A process name remains in effect until specifically changed or until the process is deleted.

2. Process names provide an identification mechanism for processes executing with the same group number. Processes can also be identified by process identifications.

For an example of the $SETPRN system service, and details on process identification and system services providing process control functions, see Section 3.5, "Process Control Services."
$SETRWM

4.65 $SETRWM - SET RESOURCE WAIT MODE

The Set Resource Wait Mode system service allows a process to indicate what action a system service should take when it lacks a system resource required for its execution:

- When resource wait mode is enabled (the default mode), the service waits until a resource is available and then resumes execution.
- When resource wait mode is disabled, the service returns control to the caller immediately with a status code indicating that a resource is unavailable.

Macro Format:

$SETRWM [watflg]

High-Level Language Format:

SYS$SETRWM([watflg])

watflg

wait indicator. A value of 0 (the default) indicates that resources are to be awaited; this is the initial setting for resource wait mode. A value of 1 indicates that failure status should be returned immediately.

Return Status:

SS$ WASCLR
- Service successfully completed. Resource wait mode was previously enabled.

SS$ WASSET
- Service successfully completed. Resource wait mode was previously disabled.

Notes:

1. The following system resources and process quotas are affected by resource wait mode:
   - System dynamic memory
   - Direct I/O quota (DIOLM)
   - Buffered I/O quota (BIOLM)
   - Buffered I/O byte count limit (BYTLM)

2. If resource wait mode is disabled, it remains disabled until it is explicitly reenabled or until the process is deleted.
$SETSFM

4.66 $SETSFM - SET SYSTEM SERVICE FAILURE EXCEPTION MODE

The Set System Service Failure Exception Mode system service controls
whether a software exception is generated when an error or severe
error status code is returned from a system service call. Initially,
system service failure exceptions are disabled; the caller should
explicitly test for successful completion following a system service
call.

Macro Format:

$SETSFM [enbflg]

High-Level Language Format:

SYS$SETSFM([enbflg])

enbflg enable indicator. A value of 1 indicates that system service
failure exceptions are to be generated. A value of 0 (the
default) disables their generation.

Return Status:

SS$ WASCLR
  Service successfully completed. Failure exceptions were
  previously disabled.

SS$ WASSET
  Service successfully completed. Failure exceptions were
  previously enabled.

Notes:

1. When enabled, system service failure exceptions are generated
   only if the service call originated from user mode. The
   $SETSFM system service can be called, however, from any
   access mode. If enabled, system service failure exception
   mode remains enabled until explicitly disabled or until the
   image exits.

2. If failure exceptions are enabled, a condition handler can be
   specified in the first longword of the procedure call stack
   or with the Set Exception Vector ($SETEXV) system service.
   If no condition handler is specified by the user, a default
   system handler is used. This condition handler causes the
   image to exit and then displays the exit status.

3. The argument list provided to the condition handler has the
code $SS_SSFFAIL in the condition name argument of the signal
array.

For an explanation and examples of condition handling routines, the
format of the argument lists passed to the condition handler, and a
discussion of the appropriate actions a condition handler may take,
see Section 3.7, "Condition Handling Services."
4.67 $SETSWM - SET PROCESS SWAP MODE

The Set Process Swap Mode system service allows a process to control whether it can be swapped out of the balance set. Once a process is locked in the balance set, it cannot be swapped out of memory until it is explicitly unlocked.

Macro Format:

$SETSWM [swpflg]

High-Level Language Format:

SYS$SETSWM([swpflg])

swpflg

Swap indicator. A value of 0 (the default) allows the process to be swapped; this is the initial setting for swap mode. A value of 1 inhibits swapping.

Return Status:

SS$ WASCLR

Service successfully completed. The process was not previously locked in the balance set.

SS$ WASET

Service successfully completed. The process was previously locked in the balance set.

SS$ NOPRIV

The process does not have the privilege to alter its swap mode.

Privilege Restrictions:

The user privilege PSWAPM is required to alter process swap mode.

Notes:

1. If a process is locked in the balance set it remains locked until explicitly unlocked or until the process is deleted.

2. Specific pages of a process's virtual address space can be locked in the balance set with the Lock Pages in Memory ($LCKPAG) system service.
$SNDACC

4.68 $SNDACC - SEND MESSAGE TO ACCOUNTING MANAGER

The Send Message to Accounting Manager system service controls accounting log activity and allows a process to write an arbitrary data message into the accounting log file.

Macro Format:

$SNDACC msgbuf ,[chan]

High-Level Language Format:

SYS$SNDACC(msgbuf ,[chan])

msgbuf
address of a character string descriptor pointing to the message buffer. The types of message and the buffer formats are described in Section 4.68.2, below.

chan
number of the channel assigned to the mailbox to receive the reply. If no channel number is specified, or if it is specified as 0 (the default), it indicates that no reply is desired.

Return Status:

SS$_NORMAL
Service successfully completed.

SS$_ACCVIO
The message buffer or buffer descriptor cannot be read by the caller.

SS$_BADPARAM
The specified message has a length of 0 or has more than 254 characters.

SS$_DEVNOTMBX
The channel specified is not assigned to a mailbox.

SS$_INSFMEM
Insufficient system dynamic memory is available to complete the service and the process has disabled resource wait mode with the Set Resource Wait Mode ($SETRWM) system service.

SS$_IVCHAN
An invalid channel number was specified, that is, a channel number of 0 or a number larger than the number of channels available.

SS$_NOPRIV
The caller does not have write access to the specified mailbox.

Privilege Restrictions:

The user privilege OPER is required to create a new log file or to enable or disable accounting.
Resources Required/Returned:

The Send Message to Accounting Manager system service requires system dynamic memory.

Note:

The general procedure for coding a call to this service involves the following steps:

1. Construct the message buffer and place its final length in the first word of the buffer descriptor.

2. Call the $SNDACC system service.

3. Check the return status code from the service to ensure successful completion.

4. Issue a read request to the mailbox specified, if any. When the read completes, check that the operation was successfully performed.

4.68.1 The Accounting Log File

By default, the system writes a record into the accounting log file whenever a job terminates. Termination records are written for interactive users, batch jobs, non-interactive processes, login failures, and print jobs. The $SNDACC system service allows users to write additional data into the accounting log and allows privileged users to disable or enable all accounting or accounting for particular types of jobs.

Table 4-5 lists the fields in the accounting record and notes which portions of the accounting record are written for each type of job.

4.68.2 Format of Messages Sent to the Accounting Manager

The $ACCDEF macro defines symbolic names for the message types, fields within the accounting record, and job type record codes for selective accounting.

A message buffer for a message to the accounting manager begins with a word defining the message type. Some message types require that data follow the message type code in the buffer. The message types and data, if any, required by each are listed below.

1. ACC$K_INSMESG

   Insert an arbitrary message in the accounting log file. The message code is followed with any arbitrary data. When the message is inserted in the accounting log file, the default header precedes the user-specified data.

2. ACC$K_NEWFILE

   Requests that the current log file be closed and a new file created. Operator privilege is required to create a new log file. No data is required for the message.
3. ACC$K_ENABACC

Enables accounting for all types of jobs. Operator privilege is required to enable accounting. No data is required for the message.

4. ACC$K_DISAACC

Disables accounting for all types of job. Operator privilege is required to disable accounting. No data is required for the message.

5. ACC$K_ENABSEL

Enables accounting for certain types of job. Operator privilege is required to selectively enable accounting. The message type code must be followed by one or more bytes indicating the type of job for which accounting is to be enabled:

<table>
<thead>
<tr>
<th>Code</th>
<th>Job Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACC$K_BATTRM</td>
<td>Batch job</td>
</tr>
<tr>
<td>ACC$K_INSMGS</td>
<td>Arbitrary messages</td>
</tr>
<tr>
<td>ACC$K_INTRRM</td>
<td>Interactive job</td>
</tr>
<tr>
<td>ACC$K_LOGTRM</td>
<td>Login failure termination</td>
</tr>
<tr>
<td>ACC$K_PRCTR</td>
<td>Non-interactive process</td>
</tr>
<tr>
<td>ACC$K_PRTJOB</td>
<td>Print job</td>
</tr>
</tbody>
</table>

The list of job type codes must be terminated with a byte containing 0.

6. ACC$K_DISASEL

Disables accounting for certain types of job. Operator privilege is required to selectively disable accounting. The message type code is followed by one or more bytes indicating the types of job for which accounting is to be disabled. The codes are listed above, under ACC$K_ENABSEL.
### Format of Accounting Log File Records

<table>
<thead>
<tr>
<th>Offset</th>
<th>Field Name</th>
<th>Length</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>ACC$W_MSGTYP</td>
<td>word</td>
<td>Record type code</td>
</tr>
<tr>
<td>2</td>
<td>ACC$W_MSGSZ</td>
<td>word</td>
<td>Length of data message</td>
</tr>
<tr>
<td>4</td>
<td>ACC$W_FINALSTS</td>
<td>longword</td>
<td>Final exit status</td>
</tr>
<tr>
<td>8</td>
<td>ACC$W_PID</td>
<td>longword</td>
<td>Process identification</td>
</tr>
<tr>
<td>12</td>
<td>ACC$W_JOBID</td>
<td>longword</td>
<td>Job identification</td>
</tr>
<tr>
<td>16</td>
<td>ACC$W_TERMTIME</td>
<td>quadword</td>
<td>System time at job termination</td>
</tr>
<tr>
<td>24</td>
<td>ACC$W_ACCOUNT</td>
<td>8 bytes</td>
<td>Account name (blank-filled)</td>
</tr>
<tr>
<td>32</td>
<td>ACC$W_USERNAME</td>
<td>12 bytes</td>
<td>User name (blank-filled)</td>
</tr>
</tbody>
</table>

#### Job Information

<table>
<thead>
<tr>
<th>Offset</th>
<th>Field Name</th>
<th>Length</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>44</td>
<td>ACC$W_CPU</td>
<td>longword</td>
<td>CPU time in 10-millisecond units</td>
</tr>
<tr>
<td>48</td>
<td>ACC$W_PAGEFNT</td>
<td>longword</td>
<td>Count of page faults during process lifetime</td>
</tr>
<tr>
<td>52</td>
<td>ACC$W_PGFNT</td>
<td>longword</td>
<td>Peak size of process paging file</td>
</tr>
<tr>
<td>56</td>
<td>ACC$W_NSPEAK</td>
<td>longword</td>
<td>Peak size of working set</td>
</tr>
<tr>
<td>60</td>
<td>ACC$W_BIOCNT</td>
<td>longword</td>
<td>Count of buffered I/O operations performed</td>
</tr>
<tr>
<td>64</td>
<td>ACC$W_DIOCNT</td>
<td>longword</td>
<td>Count of direct I/O operations performed</td>
</tr>
<tr>
<td>68</td>
<td>ACC$W_VOL</td>
<td>longword</td>
<td>Count of volumes mounted</td>
</tr>
<tr>
<td>72</td>
<td>ACC$W_OWNER</td>
<td>quadword</td>
<td>System time at login</td>
</tr>
<tr>
<td>80</td>
<td>ACC$W_USERNAME</td>
<td>constant</td>
<td>Process identification of process's owner</td>
</tr>
<tr>
<td></td>
<td>ACC$W_TERMLEN</td>
<td>constant</td>
<td>Length of non-batch job termination message</td>
</tr>
</tbody>
</table>

#### Batch Job Accounting Information

<table>
<thead>
<tr>
<th>Offset</th>
<th>Field Name</th>
<th>Length</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>84</td>
<td>ACC$W_JOB_NAME</td>
<td>8 bytes</td>
<td>Job name (blank-filled)</td>
</tr>
<tr>
<td>92</td>
<td>ACC$W_JOB_QE</td>
<td>16 bytes</td>
<td>Queue name (counted ASCII string)</td>
</tr>
<tr>
<td></td>
<td>ACC$W_JOB_LEN</td>
<td>constant</td>
<td>Length of termination record for batch jobs</td>
</tr>
</tbody>
</table>

#### Printer Job Information

<table>
<thead>
<tr>
<th>Offset</th>
<th>Field Name</th>
<th>Length</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>48</td>
<td>ACC$W_PAGCNT</td>
<td>longword</td>
<td>Symbiont page count</td>
</tr>
<tr>
<td>52</td>
<td>ACC$W_QIOCNT</td>
<td>longword</td>
<td>Symbiont QIO count</td>
</tr>
<tr>
<td>56</td>
<td>ACC$W_GETCNT</td>
<td>longword</td>
<td>Symbiont GET count</td>
</tr>
<tr>
<td>60</td>
<td>ACC$W_QTTERM</td>
<td>quadword</td>
<td>System time that job was queued</td>
</tr>
<tr>
<td>68</td>
<td>ACC$W_PRT_NAME</td>
<td>8 bytes</td>
<td>Name of print job</td>
</tr>
<tr>
<td>76</td>
<td>ACC$W_PRT_QE</td>
<td>12 bytes</td>
<td>Name of print queue</td>
</tr>
<tr>
<td></td>
<td>ACC$W_PRT_LEN</td>
<td>constant</td>
<td>Length of print job accounting record</td>
</tr>
</tbody>
</table>

#### User Data

<table>
<thead>
<tr>
<th>Offset</th>
<th>Field Name</th>
<th>Length</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>ACC$W_USER_DATA</td>
<td>132 bytes</td>
<td>User data written to accounting file</td>
</tr>
<tr>
<td></td>
<td>ACC$W_INS_LEN</td>
<td>constant</td>
<td>Length of user-written accounting file log record</td>
</tr>
</tbody>
</table>

---

1. Present in all types of log file records.
2. Present in interactive, non-interactive process, and batch job termination messages.
3. Present only in batch job termination records.
4. Present only in printer job termination records. The record contains default header record and CPU time followed by the data listed below.
5. Present in user-written messages.
4.68.3 Format of Response from Accounting Manager

If a mailbox is specified, the accounting manager returns a message in the format:

<table>
<thead>
<tr>
<th>Bits</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-15</td>
<td>MSG$_ACCRSP indicates that the message is a response from the accounting manager. (This symbolic name is defined in the $MSGDEF macro.)</td>
</tr>
<tr>
<td>16-31</td>
<td>0</td>
</tr>
<tr>
<td>32-63</td>
<td>Status code indicating the success of the operation.</td>
</tr>
</tbody>
</table>

If the mailbox cannot handle the message (either because of insufficient buffer space, or because a message is too long), or if the mailbox no longer exists when the reply is sent, the response is lost.

Status Codes Returned in the Mailbox:

SSS$_NORMAL
- Request successfully performed.

JBC$_ACMINVOP
- An invalid operation was requested.

JBC$_NOPRIV
- The process does not have the privilege to perform the requested operation.

These status codes are defined in the $JBCMSGDEF macro.
4.69 $SNDERR - SEND MESSAGE TO ERROR LOGGER

The Send Message To Error Logger system service writes an arbitrary message to the system error log file. The user-specified message is preceded by the date and time.

Macro Format:

$SNDERR msgbuf

High-Level Language Format:

SYS$SNDERR(msgbuf)

msgbuf
address of character string descriptor pointing to the message to be inserted in the system error log file.

Return Status:

SS$ NORMAL
Service successfully completed.

SS$ ACCVIO
The message buffer or buffer descriptor cannot be read by the caller.

SS$ INSFMEM
Insufficient system dynamic memory is available to complete the service and the process has disabled resource wait mode with the Set Resource Wait Mode ($SETRWM) system service.

SS$ NOPRIV
The process does not have the BUGCHK privilege.

Privilege Restrictions:

The user privilege BUGCHK is required to send a message to the error log file.

Resources Required/Returned:

The Send Message To Error Logger system service requires system dynamic memory.
4.70 **$SNDOPR** – SEND MESSAGE TO OPERATOR

The Send Message To Operator system service allows a process to send a message to one or more terminals designated as operators' terminals and optionally receive a reply.

**Macro Format:**

```
$SNDOPR msgbuf ,[chan]
```

**High-Level Language Format:**

```
SYSS$SNDOPR(msgbuf ,[chan])
```

- **msgbuf**
  - address of character string descriptor pointing to the message buffer. The types of message and the buffer formats are described in Section 4.70.1, below.

- **chan**
  - number of the channel assigned to the mailbox to which the reply is to be sent, if any. A channel number of 0 (the default) implies no mailbox unit.

**Return Status:**

- **$SS$NORMAL**
  - Service successfully completed.

- **$SS$ACCVIO**
  - The message buffer or buffer descriptor cannot be read by the caller.

- **$SS$BADPARAM**
  - The specified message has a length of 0 or has more than 128 bytes.

- **$SS$DEVNOTMBX**
  - The channel specified is not assigned to a mailbox.

- **$SS$DEVOFPLINE**
  - There is no operator designated to receive messages.

- **$SS$INSFMEM**
  - Insufficient system dynamic memory is available to complete the service and the process has disabled resource wait mode with the Set Resource Wait Mode ($SETRWM) system service.

- **$SS$IVCHAN**
  - An invalid channel number was specified; that is, a channel number of 0 or a number larger than the number of channels available.

- **$SS$NOPRIV**
  - The process does not have the privilege to send a message to the operator, the process does not have read/write access to the specified mailbox, or the channel was assigned from a more privileged access mode.
Privilege Restrictions:

The user privilege OPER is required to issue the Send Message To Operator system service to enable a terminal as an operator's terminal, reply to or cancel a user's request, or initialize the operator communication log file.

Resources Required/Returned:

The Send Message To Operator system service requires system dynamic memory.

Note:

The general procedure for using this service is as follows:

1. Construct the message buffer and place its final length in the first word of the buffer descriptor.

2. Issue the $SNDOPR system service.

3. Check the return status code from the service to ensure successful completion.

4. Issue a read request to the mailbox specified, if any. When the read completes, check that the operation was successfully performed.

4.70.1 Operator Communication

This service is used by the system to implement the REQUEST and REPLY commands, which provide communications between users and operators. An operator establishes a terminal as an operator's console by issuing the REPLY/ENABLE command, specifying the types of message that will be handled. Users can then send messages to the operator with the REQUEST command, optionally requesting replies.

Messages are displayed on a specified operator's terminal in the format:

```
Opcom -- time -- User="username" ACNT="account"
[Opcom -- *** REPLY-ID = n ***]
Opcom -- message-text
```

If a reply is requested, the operator request is kept active until the operator responds.

4.70.2 $SNDOPR Message Types and Message Formats

The $OPCDEF macro defines symbolic names for operator message types, offsets within messages, and return status codes.
The $SNDOPR system service handles five types of message:

<table>
<thead>
<tr>
<th>Code</th>
<th>Type of Request</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPC$_RQ_RQST</td>
<td>Request operator functions</td>
</tr>
<tr>
<td>OPC$_RQ_CANCEL</td>
<td>Cancel a user request</td>
</tr>
<tr>
<td>OPC$_RQ_REPLY</td>
<td>Reply to user request</td>
</tr>
<tr>
<td>OPC$_RQ_TERME</td>
<td>Enable terminal for operator's use</td>
</tr>
<tr>
<td>OPC$_RQ_LOGI</td>
<td>Initialize log file</td>
</tr>
</tbody>
</table>

Each message type has a different format. The maximum length of any message is 128 bytes, including message text.

4.70.2.1 OPC$ RQ_RQST - Constructs a message to be displayed at an operator's terminal (REQUEST command). The message format is:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPC$B_MS_TYPE</td>
<td>byte</td>
<td>OPC$ RQ_RQST identifies the type of message</td>
</tr>
<tr>
<td>OPC$B_MS_TARGET</td>
<td>3 bytes</td>
<td>Mask indicating which operators will receive the message. The symbolic names to create the mask are:</td>
</tr>
<tr>
<td>OPC$M_NM_CENTRL</td>
<td></td>
<td>Central operator</td>
</tr>
<tr>
<td>OPC$M_NM_DEVICE</td>
<td></td>
<td>Device status information</td>
</tr>
<tr>
<td>OPC$M_NM_DISK</td>
<td></td>
<td>Disk operator</td>
</tr>
<tr>
<td>OPC$M_NM_TAPE</td>
<td></td>
<td>Tape operator</td>
</tr>
<tr>
<td>OPC$M_NM_PRINT</td>
<td></td>
<td>Printer operator</td>
</tr>
<tr>
<td>OPC$M_NM_OPER1</td>
<td></td>
<td>System manager-defined operator functions</td>
</tr>
<tr>
<td>OPC$M_NM_OPER12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OPC$L_MS_RQSTID</td>
<td>Longword</td>
<td>User-specified message identification to be used for replying</td>
</tr>
<tr>
<td>OPC$L_MS_TEXT</td>
<td>0-120 bytes</td>
<td>Up to 120 bytes of message text</td>
</tr>
</tbody>
</table>

4.70.2.2 OPC$ RQ_CANCEL - Notifies an operator that a request is to be canceled.

The message format is the same as for the message type OPC$ RQ_RQST except that:

- The message type field must contain OPC$ RQCANCEL
- The message has no message text.
4.70.2.3 **OPC$ _RQ _REPLY** - Constructs a reply to a user request (REPLY command). The message format is:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPC$B _MS _TYPE</td>
<td>byte</td>
<td>OPC$ _RQ _REPLY identifies the type of message</td>
</tr>
<tr>
<td>OPC$W _MS _STATUS</td>
<td>word</td>
<td>Return status:</td>
</tr>
<tr>
<td>OPC$L _MS _RPLYID</td>
<td>longword</td>
<td>Identification of message to which reply is directed</td>
</tr>
<tr>
<td>OPC$W _MS _OUNIT</td>
<td>word</td>
<td>Unit number of terminal</td>
</tr>
<tr>
<td>OPC$T _MS _ONAME</td>
<td>--</td>
<td>Device name (counted ASCII string)</td>
</tr>
<tr>
<td>OPC$L _MS _OTEEXT</td>
<td>--</td>
<td>Reply message text, if any</td>
</tr>
</tbody>
</table>

4.70.2.4 **OPC$ _RQ _TERME** - Enables a terminal for operator use (REPLY/ENABLE command). The message format is:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPC$B _MS _TYPE</td>
<td>byte</td>
<td>OPC$ _RQ _TERME identifies the type of message</td>
</tr>
<tr>
<td>OPC$B _MS _ENABL</td>
<td>3 bytes</td>
<td>Masks defining the type of messages for which the terminal is enabled</td>
</tr>
<tr>
<td>OPC$L _MS _MASK</td>
<td>longword</td>
<td></td>
</tr>
<tr>
<td>OPC$W _MS _OUNIT</td>
<td>word</td>
<td>Unit number of terminal</td>
</tr>
<tr>
<td>OPC$T _MS _ONAME</td>
<td>--</td>
<td>Device name (counted ASCII string)</td>
</tr>
</tbody>
</table>

4.70.2.5 **OPC$ _RQ _LOGI** - Initializes the log file of operator messages (REPLY/LOG command). The message format is:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPC$B _MS _TYPE</td>
<td>byte</td>
<td>OPC$ _RQ _LOGI identifies the type of message</td>
</tr>
<tr>
<td>--</td>
<td>7 bytes</td>
<td>Ignored</td>
</tr>
<tr>
<td>OPC$W _MS _OUNIT</td>
<td>word</td>
<td>Unit number of terminal</td>
</tr>
<tr>
<td>OPC$T _MS _ONAME</td>
<td>--</td>
<td>Device name (counted ASCII string)</td>
</tr>
</tbody>
</table>
4.70.3 Format of Response from Operator Communication Manager

When the operator replies to a message, the reply is placed in the specified mailbox in the format:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPC$B_MS_TYPE</td>
<td>word</td>
<td>MSG$OPREPLY indicates that the message is a response to an operator's request. This symbolic name is defined in the $MSGDEF macro.</td>
</tr>
<tr>
<td>OPC$W_MS_STATUS</td>
<td>word</td>
<td>Return status.</td>
</tr>
<tr>
<td>OPC$L_MS_RPLYID</td>
<td>longword</td>
<td>Identification of message for which reply is made (specified in user request message)</td>
</tr>
<tr>
<td>OPC$L_MS_TEXT</td>
<td>0-128 bytes</td>
<td>Up to 128 bytes of message text taken from reply</td>
</tr>
</tbody>
</table>

If the mailbox specified to receive the reply cannot handle the reply message (either because of insufficient buffer space or because the message is too big), the message is lost.

Status Codes Returned in Mailbox:

OPCS_NOPERATOR
- Success. There was no operator enabled to receive the message.

OPCS_RQSTCMPLTE
- Success. The operator completed the request.

OPCS_RQSTPEND
- Success. The operator will perform the request when possible.

OPCS_RQSTABORT
- The operator could not satisfy the request.

OPCS_RQSTCAN
- The caller canceled the request.
4.71 $SNDSMB - SEND MESSAGE TO SYMBIONT MANAGER

The Send Message To Symbiont Manager system service is used by the operating system to queue user's print files to a system printer or to queue command procedure files for detached job execution.

Symbiont manager requests:

- Create and delete queues
- Add or delete files from a queue
- Change the attributes of files in a queue
- Start and restart dequeuing

Macro Format:

$SNDSMB msgbuf ,[chan]

High-Level Language Format:

SYS$SNDSMB(msgbuf ,[chan])

msgbuf
address of a character string descriptor pointing to the message buffer. The buffer formats and the types of messages are described in Section 4.71.1, below.

chan
number of the channel assigned to the mailbox to receive the reply. If no channel number is specified, or if it is specified as 0 (the default), it indicates that no reply is desired.

Return Status:

SS$ NORMAL
- Service successfully completed.

SS$ ACCVIO
- The message buffer or buffer descriptor cannot be read by the caller.

SS$ BADPARAM
- The specified message has a length of 0 or has more than 200 characters.

SS$ DEVNOTMBX
- The specified channel is not assigned to a mailbox.

SS$ INSFMEM
- Insufficient system dynamic memory is available to complete the service and the process has disabled resource wait mode with the Set Resource Wait Mode ($SETRWM) system service.
SYSTEM SERVICE DESCRIPTIONS
$SNDNSMB - SEND MESSAGE TO SYMBIONT MANAGER

SS$_IVCHAN
An invalid channel number was specified; that is, a channel number of 0 or a number larger than the number of channels available.

SS$_NOPRIV
The caller does not have write access to the specified mailbox.

Resources Required/Returned:
The Send Message To Symbiont Manager system service requires system dynamic memory.

Privilege Restrictions:
There are two levels of privilege involved in symbiont control:

- The user privilege OPER is required to manipulate device queues, to modify job queues for other users, or increase the priority of a job within a queue.
- A process can manipulate any jobs owned by processes in its group.

Note:
The general procedure for using this service is as follows:
1. Construct the message buffer and place its final length in the first word of the buffer descriptor.
2. Issue the $SNDNSMB system service.
3. Check the return status code from the service to ensure successful completion.
4. Issue a read request to the mailbox specified, if any. When the read completes, check that the operation was successfully performed.

4.71.1 Format of Messages Sent to Symbiont Manager

Messages are variable-length, and their formats depend on the request type. Each request type can require from 0 to 5 additional data fields, and can be followed by options. Some options require additional data.

The general message format is:
request[queue][devname][fileid][dirname]
[fname][jobid][jobname][option[opdata]]

request
16-bit field indicating the request type. The $SMRDEF macro defines symbolic codes for each request in the format:

SMR$C_code

Valid request codes, and required and optional fields for fields for each, are listed in Table 4-6.
queue name
16-byte queue name. The length of the name must be in the first byte. A queue name can be a physical device name (for example, LPA0:], a logical name (for example, SYS$PRINT), or a designated name string, such as BATCH or AFTER5.

Some request types require two queue names, for example SMRSK_MERGE.

devname
16-byte field containing the name of the device on which the file resides. The length of the device name must be in the first byte. The device name is returned by RMS as a counted ASCII string in the NAM$T_DVI field of the auxiliary name block (NAM) when the file is opened.

fileid
6-byte file identification. RMS returns the file identification in the auxiliary name block (NAM) beginning at the offset NAM$W_FID when the file is opened.

dirname
6-byte directory name returned by RMS in the name block (NAM) at the offset NAM$W_DID.

filename
20-byte field containing the name of a file to be queued. The first byte in the field must contain the length.

jobid
16-bit job header identifying the job. The jobid is returned in the message queued to the mailbox on completion of the operation.

jobname
8-byte blank-filled ASCII name string.

option
byte indicating an optional parameter for the request. The $SMRDEF macro defines symbolic names for the options in the format:

SMOSC_option

Valid options for each request type are listed in Table 4-6. The options, and any data required by each, are listed in Table 4-7.

opdata
any data required by the specified option.

Syntax Notes:
1. Fields within the message buffer must be placed in consecutive positions in the buffer, with no intervening blanks.

2. The message length passed to the service indicates the total length of the buffer. If a byte of binary 0's follows an option or its required data, the message scan is terminated. Therefore, fixed-length message buffers can be used, with a 0 indicating termination of the option list.
SYSTEM SERVICE DESCRIPTIONS
$SNDMSB - SEND MESSAGE TO SYMBIONT MANAGER

The following example shows an input message buffer for the $SNDMSB system service:

ADDLIST:
  .WORD  SMRK_ADDFIL  #MESSAGE BUFFER
  .BLKB  16      #REQUEST TYPE TO ADD A FILE
  .BLKB  3     #MOVE DEVICE NAME HERE (COUNTED STRING)
  .BLKB  20   #FILEID HERE
  .BLKB  20    #MOVE FILENAME HERE
  .BLKB  10   #LEAVE ROOM FOR 10 OPTIONS
  .LONG ADDDESC-ADDLIST  #DESCRIPTION FOR MESSAGE
  .LONG ADDLIST    #ADDRESS OF BUFFER

$SNDMSB $ MSGBUF=ADDDESC #ADD FILE TO QUEUE
<table>
<thead>
<tr>
<th>Request</th>
<th>Function</th>
<th>Required Data</th>
<th>Valid Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMR$K_ABORT</td>
<td>Stops printing the current file and skips to the next file</td>
<td>queueename</td>
<td>SMR$K_REQUEUE</td>
</tr>
<tr>
<td>SMR$K_ADDFIL</td>
<td>Adds a file to a job</td>
<td>devname, field 1, dirname 1, filename 2</td>
<td>SMR$K_COPIES, SMR$K_BRSTPAG, SMR$K_DELETE, SMR$K_DOUBLE, SMR$K_FLAGPAG, SMR$K_NOBRSTPAG, SMR$K_NOCOPY, SMR$K_PACNT, SMR$K_PAGHDR</td>
</tr>
<tr>
<td>SMR$K_ALTER</td>
<td>Changes attributes of a previously queued job and requeues the job</td>
<td>queueename, jobid</td>
<td>SMR$K_FORMTP, SMR$K_HOLD, SMR$K_JOCOPY, SMR$K_JONAME, SMR$K_JOBPRI, SMR$K_LOWER, SMR$K_NOLOWER, SMR$K_RSLTIM</td>
</tr>
<tr>
<td>SMR$K_ASSIGN</td>
<td>Directs a queue to a specific device</td>
<td>queueename [devname]</td>
<td>None</td>
</tr>
<tr>
<td>SMR$K_CLSJOB</td>
<td>Closes the job</td>
<td>None</td>
<td>SMR$K_FORMTP, SMR$K_HOLD, SMR$K_JOBPRI, SMR$K_RSLTIM</td>
</tr>
<tr>
<td>SMR$K_CREJOB</td>
<td>Creates a job</td>
<td>queueename</td>
<td>SMR$K_FORMTP, SMR$K_HOLD, SMR$K_JOCOPY, SMR$K_JOBPRI, SMR$K_LOWER, SMR$K_NOLOWER, SMR$K_PARAMS, SMR$K_RSLTIM</td>
</tr>
<tr>
<td>SMR$K_DELETE</td>
<td>Deletes a device queue</td>
<td>queueename</td>
<td>None</td>
</tr>
<tr>
<td>SMR$K_ENTER</td>
<td>Enters a file in a queue for a device</td>
<td>queueename, devname, field 1, dirname 1, filename 2</td>
<td>SMR$K_BRSTPAG, SMR$K_COPIES, SMR$K_DELETE, SMR$K_DOUBLE, SMR$K_FLAGPAG, SMR$K_FORMTP, SMR$K_HOLD, SMR$K_JOCOPY, SMR$K_LOWER, SMR$K_NOBRSTPAG, SMR$K_NO?EED, SMR$K_NOFLAGPAG, SMR$K_NOLOWER, SMR$K_PACNT, SMR$K_PAGHDR, SMR$K_JOBPRI, SMR$K_RSLTIM</td>
</tr>
</tbody>
</table>

1 The dirname field is required only if file is to be deleted after processing.
2 The filename field is optional; it can be used for informational purposes.
## System Service Descriptions

**$SND$SMB - Send Message to Symbiont Manager**

Table 4-6 (Cont.)

### Request Types for Symbiont Manager Messages

<table>
<thead>
<tr>
<th>Request</th>
<th>Function</th>
<th>Required Data</th>
<th>Valid Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMR$K$ INITIAL</td>
<td>Initializes or reinitializes a queue</td>
<td>queueename</td>
<td>SMOK$K CURFORM, SMOK$K DEFBRT, SMOK$K DEFFLAG, SMOK$K DEJOB, SMOK$K DSNAP,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SMOK$K GENDEV, SMOK$K GENFRT, SMOK$K INIPRI, SMOK$K JOBLIM, SMOK$K NODEFBRT,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SMOK$K NODEFFLAG, SMOK$K NGENDEV, SMOK$K NOGENFRT, SMOK$K NORMDEV, SMOK$K</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>TRDEV</td>
</tr>
<tr>
<td>SMR$K$ JUSTIFY</td>
<td>Issues hardware form feed</td>
<td>queueename</td>
<td>None</td>
</tr>
<tr>
<td>SMR$K$ MERGE</td>
<td>Deletes jobs from second queue and places them in first</td>
<td>queueename&lt;sup&gt;1&lt;/sup&gt;, queueename&lt;sup&gt;2&lt;/sup&gt;</td>
<td>None</td>
</tr>
<tr>
<td>SMR$K$ PAUSE</td>
<td>Temporarily suspends current operation</td>
<td>queueename</td>
<td>None</td>
</tr>
<tr>
<td>SMR$K$ REDIRECT</td>
<td>Redirects second queue to first queue</td>
<td>queueename&lt;sup&gt;1&lt;/sup&gt;, queueename&lt;sup&gt;2&lt;/sup&gt;</td>
<td>None</td>
</tr>
<tr>
<td>SMR$K$ RELEASE</td>
<td>Releases a held job for printing</td>
<td>queueename jobid&lt;sup&gt;3&lt;/sup&gt;</td>
<td>None</td>
</tr>
<tr>
<td>SMR$K$ RMVJOB</td>
<td>Removes a job from a queue</td>
<td>jobid</td>
<td>None</td>
</tr>
<tr>
<td>SMR$K$ START</td>
<td>Enables printing on a device, resumes printing on a paused</td>
<td>queueename</td>
<td>SMOK$K CURFORM, SMOK$K DEFBRT, SMOK$K DEFFLAG, SMOK$K DEJOB, SMOK$K GENDEV,</td>
</tr>
<tr>
<td>SMR$K$ STOP</td>
<td>Stops printing on a device (for a batch job, equivalent to</td>
<td>queueename</td>
<td>SMOK$K GENFRT, SMOK$K NGENDEV, SMOK$K NODEFBRT, SMOK$K NODEFFLAG, SMOK$K</td>
</tr>
<tr>
<td>SMR$K$ SYNCJOB</td>
<td>Waits for a batch job to complete</td>
<td>queueename jobid&lt;sup&gt;4&lt;/sup&gt;, queueename jobname</td>
<td>SMOK$K NORMDEV, SMOK$K NGENFRT, SMOK$K TRDEV</td>
</tr>
</tbody>
</table>

<sup>1</sup> The dirname field required only if file is to be deleted after processing.

<sup>2</sup> The filename field is optional; it can be used for informational purposes.

<sup>3</sup> A jobid is optional; if specified as 0 or not specified, the first job in queue is released.

<sup>4</sup> Either the jobid or the jobname must be specified.

4-164
<table>
<thead>
<tr>
<th>Option</th>
<th>Function</th>
<th>Required Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMO$K_BRSTPAG</td>
<td>Specifies that a burst page should be printed</td>
<td>None</td>
</tr>
<tr>
<td>SMO$K_COPIES</td>
<td>Specifies the number of copies of the file to print</td>
<td>Number of copies (1 byte)</td>
</tr>
<tr>
<td>SMO$K_CURFORM</td>
<td>Defines form type currently on printer</td>
<td>Type of form (1 byte)</td>
</tr>
<tr>
<td>SMO$K_DEFBRST</td>
<td>Specifies that queue prints burst page by default</td>
<td>None</td>
</tr>
<tr>
<td>SMO$K_DEFFLAG</td>
<td>Specifies that queue prints flag page by default</td>
<td>None</td>
</tr>
<tr>
<td>SMO$K_DELETE</td>
<td>Deletes file after printing</td>
<td>None</td>
</tr>
<tr>
<td>SMO$K_DETJOB</td>
<td>Defines queue as a detached job (batch) queue</td>
<td>None</td>
</tr>
<tr>
<td>SMO$K_DISWAP</td>
<td>Disables swapping of all batch jobs in queue</td>
<td>None</td>
</tr>
<tr>
<td>SMO$K_DOUBLE</td>
<td>Double-spaces printer output</td>
<td>None</td>
</tr>
<tr>
<td>SMO$K_FLAGPAG</td>
<td>Specifies that a flag page should be printed</td>
<td>None</td>
</tr>
<tr>
<td>SMO$K_FORMTYPE</td>
<td>Specifies the form type</td>
<td>Type of form (1 byte)</td>
</tr>
<tr>
<td>SMO$K_GENDEV</td>
<td>Defines the queue as a generic device queue</td>
<td>None</td>
</tr>
<tr>
<td>SMO$K_GENPRT</td>
<td>Defines the queue as a generic printer file queue</td>
<td>None</td>
</tr>
<tr>
<td>SMO$K_HOLD</td>
<td>Holds job until specifically released</td>
<td>None</td>
</tr>
<tr>
<td>SMO$K_INIPRI</td>
<td>Specifies initial priority of batch job</td>
<td>Priority (1 byte) range: 0 through 15</td>
</tr>
<tr>
<td>SMO$K_JOBCOPY</td>
<td>Specifies a repeat count for the entire job</td>
<td>Repeat count (1 byte)</td>
</tr>
<tr>
<td>SMO$K_JOBLIM</td>
<td>Specifies maximum number of jobs in batch queue</td>
<td>Number of jobs (1 byte)</td>
</tr>
<tr>
<td>SMO$K_JOBNAME</td>
<td>Specifies the job name</td>
<td>Counted ASCII string (1 to 8 bytes)</td>
</tr>
<tr>
<td>SMO$K_JOBPRI</td>
<td>Specifies priority for queuing of a job</td>
<td>Priority (1 byte) range: 0 through 31</td>
</tr>
<tr>
<td>SMO$K_LOWER</td>
<td>Specifies that printer must be equipped with uppercase and lowercase characters</td>
<td>None</td>
</tr>
<tr>
<td>SMO$K_NEXTJOB</td>
<td>Terminates current job and start printing with next job</td>
<td>None</td>
</tr>
<tr>
<td>Option</td>
<td>Function</td>
<td>Required Data</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------------------------------------------------------------------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>SMO$K_NOBRSTPAG</td>
<td>Specifies that no burst page should be printed</td>
<td>None</td>
</tr>
<tr>
<td>SMO$K_NODEFBRST</td>
<td>Specifies that printer does not generate burst page by default</td>
<td>None</td>
</tr>
<tr>
<td>SMO$K_NODEFFLAG</td>
<td>Specifies that printer does not generate flag page by default</td>
<td>None</td>
</tr>
<tr>
<td>SMO$K_NOFEED</td>
<td>Cancels automatic form feed for output</td>
<td>None</td>
</tr>
<tr>
<td>SMO$K_NOPLAPAG</td>
<td>Specifies that no flag page should be printed</td>
<td>None</td>
</tr>
<tr>
<td>SMO$K_NOGENDEV</td>
<td>Disallows generic spooling to the device</td>
<td>Number of pages (1 word)</td>
</tr>
<tr>
<td>SMO$K_NOGENPRT</td>
<td>Disallows generic printing on the specified device</td>
<td>None</td>
</tr>
<tr>
<td>SMO$K_NOLOWER</td>
<td>Specifies that lowercase printer is not required</td>
<td>None</td>
</tr>
<tr>
<td>SMO$K_NOTRMDEV</td>
<td>Specifies that device is not a terminal</td>
<td>None</td>
</tr>
<tr>
<td>SMO$K_PAGCNT</td>
<td>Specifies the number of pages to print</td>
<td>None</td>
</tr>
<tr>
<td>SMO$K_PAGHDR</td>
<td>Prints file specification on the top of each output page</td>
<td>One or more counted ASCII strings terminated by 0 (maximum length of all strings is 63 bytes)</td>
</tr>
<tr>
<td>SMO$K_PARAMS</td>
<td>Specifies parameters for a batch job</td>
<td>None</td>
</tr>
<tr>
<td>SMO$K_REQUEUE</td>
<td>Places aborted line printer job back into the queue</td>
<td>Binary absolute time value (quadword)</td>
</tr>
<tr>
<td>SMO$K_RLSTIM</td>
<td>Specifies time to release a held job</td>
<td>Signed 16-bit integer specifying plus or minus page count</td>
</tr>
<tr>
<td>SMO$K_SPCCNT</td>
<td>Restarts current job backspacing or forward spacing pages</td>
<td>None</td>
</tr>
<tr>
<td>SMO$K_TOPOFILE</td>
<td>Restarts current job at top of file</td>
<td>None</td>
</tr>
<tr>
<td>SMO$K_TRMDEV</td>
<td>Specifies that device is a terminal</td>
<td>None</td>
</tr>
</tbody>
</table>
4.71.2 Format of Response from Symbiont Manager

If a mailbox is specified, the symbiont manager returns to it the following information:

<table>
<thead>
<tr>
<th>Bits</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-15</td>
<td>MSG$_SMBRSP indicates that the message is from the symbiont manager. (This name is defined in the $MSGDEF macro.)</td>
</tr>
<tr>
<td>16-31</td>
<td>jobid</td>
</tr>
<tr>
<td>32-63</td>
<td>status code indicating the success of the operation.</td>
</tr>
</tbody>
</table>

If the mailbox cannot handle the message (either because of insufficient buffer space, or because a message is too long), or if the mailbox no longer exists when the reply is sent, the response is lost.

Status Codes Returned in Mailbox:

JBC$_NORMAL
- Service successfully completed.

JBC$_ILLDEVNAM
- The device name specified has more than 15 characters.

JBC$_ILLDEVTYP
- The symbiont manager cannot create a queue for the device type specified.

JBC$_ILLPILNAM
- The filename specified has more than 19 characters.

JBC$_INVREQ
- An invalid request type was specified.

JBC$_NOOPENJOB
- There is no outstanding open print job for the caller.

JBC$_NOPRIV
- The process does not have the privilege to perform the requested operation.

JBC$_NOQUEHDR
- The symbiont manager has no more space to allocate a queue header.

JBC$_NOQUESPACE
- The specified device queue is full.

JBC$_NOSUCHJOB
- The specified record was not a print job.

JBC$_NOSUCHQUE
- There is no queue for the specified device.

JBC$_QUENOSTOP
- The specified queue is still active.
JBC$_SMINVOPR
  The request type specified is illegal; or, an attempt was made to start a queue that was already started.

JBC$_SMINVOPT
  A specified option is invalid for the request type.

JBC$_SMINVREQ
  An invalid request type was specified.

JBC$_SMZEROJOB
  A job was released that had no files in it.

JBC$_SYMBDSAB
  The symbiont manager is disabled.

These status codes are defined in the $JBCMSGDEF macro.
4.72 $SUSPND - SUSPEND PROCESS

The Suspend Process system service allows a process to suspend itself or another process. A suspended process cannot receive ASTs or otherwise be executed until another process resumes or deletes it.

Macro Format:

$SUSPND [pidadr], [prcnam]

High-Level Language Format:

SYS$SUSPND([pidadr], [prcnam])

pidadr
address of a longword containing the process identification of the process to be suspended.

prcnam
address of a character string descriptor pointing to the 1- to 15-character process name string. The process name is implicitly qualified by the group number of the process issuing the suspend.

If neither a process identification nor a process name is specified, the caller is suspended. For details on how the service interprets the PIDADR and PRCNAM arguments, see Table 3-3. Table 3-3 is in Section 3.5 "Process Control Services."

Return Status:

SS$NORMAL
Service successfully completed.

SS$ACCVIO
The process name string or string descriptor cannot be read, or process identification cannot be written, by the caller.

SS$INSMEM
Insufficient system dynamic memory is available to complete the service and the process has disabled resource wait mode with the Set Resource Wait Mode ($SETRWM) system service.

SS$IVLOGNAM
The specified process name has a length of 0, or has more than 15 characters.

SS$NONEPR
Warning. The specified process does not exist, or an invalid process identification was specified.

SS$NOPRIV
The target process was not created by the caller and the requesting process does not have group or world process control privilege.
SYSTEM SERVICE DESCRIPTIONS
$SUSPEND - SUSPEND PROCESS

Privilege Restrictions:

User privileges are required to suspend:

- Other processes in the same group (GROUP privilege)
- Any other process in the system (WORLD privilege)

Resources Required/Returned:

The Suspend Process system service requires system dynamic memory.

Notes:

1. The suspend process system service completes successfully if the target process is already suspended.

2. Unless it has pages locked in the balance set, a suspended process can be removed from the balance set to allow other processes to execute.

3. The Resume Process ($RESUME) system service allows a suspended process to continue. If one or more resume requests are issued for a process that is not suspended, a subsequent suspend request completes immediately, that is, the process is not suspended. No count is maintained of outstanding resume requests.

For more information on process suspension, see Section 3.5.5, "Process Hibernation and Suspension."
4.73 $TRNLOG - TRANSLATE LOGICAL NAME

The Translate Logical Name system service searches the logical name tables for a specified logical name and returns an equivalence name string. The process, group, and system logical name tables are searched in that order.

The first string match returns the equivalence string into a user-specified buffer; the search is not recursive.

Macro Format:

$TRNLOG lognam,[rslllen],rslbuf,[table],[acmode],[dsbmsk]

High-Level Language Format:

SYS$TRNLOG(lognam,[rslllen],rslbuf,[table],[acmode],[dstmsk])

lognam
address of a character string descriptor pointing to the logical name string.

rslllen
address of a word to receive the length of the translated equivalence name string.

rslbuf
address of a character string descriptor pointing to the buffer which is to receive the resultant equivalence name string.

table
address of a byte to receive the number of the logical name table in which the match was found. A return value of 0 indicates that the logical name was found in the system logical name table; 1 indicates the group table, and 2 indicates the process table.

acmode
address of a byte to receive the access mode from which the logical name table entry was made. Data received in this byte is valid only if the logical name match was found in table 2, the process logical name table.

dsbmsk
mask in which bits set to 1 disable the search of particular logical name tables. If bit 0 is set, the system logical name table is not searched; if bit 1 is set, the group logical name table is not searched; if bit 2 is set, the process logical name table is not searched.

If no mask is specified, or is specified as 0 (the default), all three logical name tables are searched.
Return Status:

SS$ _NORMAL
   Service successfully completed. The equivalence name string was placed in the output buffer.

SS$ _NOTRAN
   Service successfully completed. The input logical name string was placed in the output buffer because no equivalence name was found.

SS$ _ACCVIO
   The logical name string or string descriptor cannot be read, or the output length, output buffer, or table or access mode field cannot be written, by the caller.

SS$ _IVLOGNAM
   The specified logical name string has a length of 0 or has more than 63 characters.

SS$ _RESULTOVF
   The buffer to receive the resultant string has a length of zero, or it is smaller than the string.

Note:

If the first character of a specified logical name is an underline character (_), no translation is performed. However, the underscore character is removed from the string and the modified string is returned in the output buffer.

For an example of the $TRNLOG system service, see Section 3.3, "Logical Name Services."
4.74 $ULKPAG - UNLOCK PAGES FROM MEMORY

The Unlock Pages from Memory system service releases the page lock on
a page or range of pages previously locked in memory by the Lock Pages
in Memory ($LCKPAG) service.

Macro Format:

$ULKPAG inadr ,[retadr] ,[acmode]

High-Level Language Format:

SYS$ULKPAG(inadr ,[retadr] ,[acmode])

inadr
address of a 2-longword array containing the starting and ending
virtual addresses of the pages to be unlocked. If the starting
and ending virtual addresses are the same, a single page is
unlocked. Only the virtual page number portion of the virtual
addresses is used; the low-order 9 bits are ignored.

retadr
address of a 2-longword array to receive the starting and ending
virtual addresses of the pages actually unlocked.

acmode
access mode of the locked pages. The specified access mode is
maximized with the access mode of the caller. The resultant
access mode must be equal to or more privileged than the access
mode of the owner of each page in order to unlock the page.

Return Status:

SS$ WASCLR
Service successfully completed. At least one of the specified
pages was previously unlocked.

SS$ WASSET
Service successfully completed. All of the specified pages were
previously locked.

SS$ ACCVIO
1. The input array cannot be read, or the output array cannot be
   written, by the caller.
2. A page in the specified range is inaccessible or does not
   exist.

Privilege Restrictions:
1. The user privilege PSWAPM is required to lock or unlock pages
   from memory.
2. The access mode of the caller must be equal to or more
   privileged than the access mode of the owner of the pages
   that are to be unlocked.
Notes:

1. If more than one page is being unlocked and it is necessary to determine specifically which pages had been previously unlocked, the pages should be unlocked one at a time.

2. If an error occurs while multiple pages are being unlocked, the return array, if requested, indicates the pages that were successfully unlocked before the error occurred. If no pages were unlocked, both longwords of the return address array contain a -1.

3. Locked pages are automatically unlocked at image exit, when the system deletes the pages.
4.75 $ULWSET - UNLOCK PAGES FROM WORKING SET

The Unlock Pages from Working Set system service allows a process to specify that a group of pages that were previously locked in the working set are to be unlocked and become candidates for page replacement like other working set pages.

Macro Format:

$ULWSET inadr ,[retadr] ,[acmode]

High-Level Language Format:

SYS$ULWSET(inadr ,[retadr] ,[acmode])

inadr
address of a 2-longword array containing the starting and ending virtual addresses of the pages to be unlocked. If the starting and ending virtual address are the same, a single page is unlocked. Only the virtual page number portion of the virtual addresses is used; the low-order 9 bits are ignored.

retadr
address of a 2-longword array to receive the starting and ending virtual addresses of the pages actually unlocked.

acmode
access mode on behalf of which the request is being made. The specified access mode is maximized with the access mode of the caller. The resultant access mode must be equal to or more privileged than the access mode of the owner of each page in order to unlock the page.

Return Status:

SS$_WASCLR
Service successfully completed. At least one of the specified pages was previously unlocked.

SS$_WASSET
Service successfully completed. All of the specified pages were previously locked in the working set.

SS$_ACCVIO
1. The input array cannot be read, or the output array cannot be written, by the caller.
2. A page in the specified range is inaccessible or does not exist.

SS$_NOPRIV
A page in the specified range is in the system address space.
Privilege Restriction:

The access mode of the caller must be equal to or more privileged than the access mode of the owner of the pages that are to be unlocked.

Notes:

1. If more than one page is being unlocked and it is necessary to determine specifically which pages had been previously unlocked, the pages should be unlocked one at a time.

2. If an error occurs while multiple pages are being unlocked, the return array, if requested, indicates the pages that were successfully unlocked before the error occurred. If no pages were unlocked, both longwords in the return address array contain a -1.
4.76 $UNWIND - UNWIND CALL STACK

The Unwind Call Stack system service allows a condition handling routine to unwind the procedure call stack to a specified depth. Optionally, a new return address can be specified to alter the flow of execution when the topmost call frame has been unwound.

Macro Format:

$UNWIND [depadr],[newpc]

High-Level Language Format:

SYS$UNWIND([depadr],[newpc])

depadr
address of a longword indicating the depth to which the stack is to be unwound. A depth of 0 indicates the call frame that was active when the condition occurred, 1 indicates the caller of that frame, 2 indicates the caller of the caller of the frame, and so on. If depth is specified as 0 or less, no unwind occurs; a successful status code is returned. If no address is specified, the unwind is performed to the caller of the frame that established the condition handler.

newpc
address to be given control when the unwind is complete.

Return Status:

SS$ NORMAL
Service successfully completed.

SS$ ACCVIO
The call stack is not accessible to the caller. This condition is detected when the call stack is scanned to modify the return address.

SS$ INSFRAME
There are insufficient call frames to unwind to the specified depth.

SS$ NOSIGNAL
Warning. No signal is currently active for an exception condition.

SS$ UNWINDING
Warning. An unwind is already in progress.
Note:

The actual unwind is not performed immediately. Rather, the return addresses in the call stack are modified so that when the condition handler returns, the unwind procedure is called from each frame that is being unwound.

For an explanation of condition handling and an example of a call to $UNWIND, see Section 3.7, "Condition Handling Services."
4.77 $SUPDSEC - UPDATE SECTION FILE ON DISK

The Update Section File on Disk system service writes all modified pages in an active private or global section back into the section file on disk. One or more I/O requests are queued, based on the number of pages that have been modified.

Macro Format:

```
$SUPDSEC inadr , [retadr], [acmode], [updflg], [efn], [iosb],
[astadr], [astprm]
```

High-Level Language Format:

```
SYSSUPDSEC(inadr , [retadr], [acmode], [updflg], [efn], [iosb],
[astadr], [astprm])
```

**inadr**
address of a 2-longword array containing the starting and ending virtual addresses of the pages to be potentially written back into the section file. The $SUPDSEC system service locates pages within this range that were modified and writes only the modified pages (with contiguous pages, if convenient) back into the section file on disk.

If the starting and ending virtual addresses are the same, a single page is a candidate for writing. Only the virtual page number portion of the virtual addresses is used; the low-order 9 bits are ignored.

**retadr**
address of a 2-longword array to receive the starting and ending virtual addresses of the first and last pages queued for writing in the first I/O request.

**acmode**
access mode on behalf of which the service is performed. The specified access mode is maximized with the access mode of the caller. The resultant access mode is used to determine whether the caller can actually write the pages.

**updflg**
update indicator for read/write global sections. If specified as 0 (the default), all read/write pages in the global section are updated in the section file on disk, regardless of whether or not they have been modified. If specified as 1, it indicates that the caller is the only process that is actually writing the global section, and that only those pages that were actually modified by the caller are to be written.

**efn**
number of an event flag to set when the section file is updated. If not specified, it defaults to 0.

**iosb**
address of a quadword I/O status block that is to receive the completion status when the section file has been updated.
SYSTEM SERVICE DESCRIPTIONS
$UPDSEC - UPDATE SECTION FILE ON DISK

astadr
address of the entry mask of an AST service routine to be executed when the section file has been updated. If specified, the AST service routine executes at the access mode from which the section file update was requested.

astprm
AST parameter to be passed to the AST service routine.

Return Status:

SS$ _NORMAL
Service successfully completed. One or more I/O requests were queued.

SS$ _NOTMODIFIED
Service successfully completed. No pages in the input address range were section pages that had been modified; no I/O requests were queued.

SS$ _ACCVIO
The input address array cannot be read, or the output address array cannot be written, by the caller.

SS$ _EXQUOTA
The process has exceeded its AST limit quota.

SS$ _ILLEFC
An illegal event flag number was specified.

SS$ _IVSECFLG
An invalid flag was specified.

SS$ _NOPRIV
A page in the specified range is in the system address space.

SS$ _PAGOWNVIO
A page in the specified range is owned by an access mode more privileged than the access mode of the caller.

SS$ _UNASEFC
The process is not associated with the cluster containing the specified event flag.

Privilege Restrictions:

Only pages that are owned by the calling or a less privileged access mode can be updated.

Resources Required/Returned:

The Update Section File on Disk system service requires the process's direct I/O limit (DIRIO) to queue the I/O request; and, if the ASTADR argument is specified, the process's AST limit quota (ASTLM).
SYSTEM SERVICE DESCRIPTIONS
$UPDSEC - UPDATE SECTION FILE ON DISK

Notes:

1. The $UPDSEC system service scans pages starting at the address contained in the first longword of the location pointed to by the INADR argument and ending with the address in the second longword. Within this range, pages are candidates for being updated based on whether they are read/write pages that were modified. Unmodified pages that share a cluster with modified pages are also written. The ending address can be lower than the starting address.

2. If the $UPDSEC system service returns an error, both longwords in the return address array contain a -1. In this case, no I/O completion is indicated, that is, the even flag is not set, no AST is delivered, and the I/O status block is not posted.

3. Proper use of this service requires the caller to synchronize completion of the update request by checking the return status from $UPDSEC. If $SS$ NOTMODIFIED is returned, the caller can continue. If $SS$ NORMAL is returned, the caller should wait for the I/O to complete and then check the status returned in the I/O status block.

When all I/O is complete, the I/O status block, if specified, is filled in as follows:

1. The first word contains the completion status of the output request.

2. If an error occurred in the I/O request, the first bit in the second word is set if a hardware write error occurred.

3. The second longword contains the virtual address of the first page that was not written.
$WAITFR

4.78  $WAITFR – WAIT FOR SINGLE EVENT FLAG

The Wait for Single Event Flag system service tests a specific event flag and returns immediately if the flag is set. Otherwise, the process is placed in a wait state until the event flag is set.

Macro Format:

$WAITFR efn

High-Level Language Format:

SYS$WAITFR(efn)

efn  number of the event flag for which to wait.

Return Status:

SS$ NORMAL
  Service successfully completed.

SS$ ILLEFC
  An illegal event flag number was specified.

SS$ UNASEFC
  The process is not associated with the cluster containing the specified event flag.

Note:

The wait state caused by this service can be interrupted by an asynchronous system trap (AST) if (1) the access mode at which the AST executes is less than or equal to the access mode from which the wait was issued and (2) the process is enabled for ASTs at that access mode.

When the AST service routine completes execution, the system repeats the $WAITFR request. If the event flag has been set, the process resumes execution.
4.79 $WAKE - WAKE

The Wake system service activates a process that has placed itself in a state of hibernation with the Hibernate ($HIBER) system service.

Macro Format:

$WAKE [pidadr], [prcnam]

High-Level Language Format:

SYS$WAKE([pidadr], [prcnam])

**pidadr**
address of a longword containing the process identification of the process to be awakened.

**prcnam**
address of a character string descriptor pointing to the process name string. The name is implicitly qualified by the group number of the process issuing the wake.

If neither a process identification nor a process name is specified, the wake request is for the caller. For details on how the service interprets the PIDADR and PRCNAM arguments, see Table 3-3. Table 3-3 is in Section 3.5, "Process Control Services."

Return Status:

**SS$ NORMAL**
Service successfully completed.

**SS$ ACCVIO**
The process name string or string descriptor cannot be read, or the process identification cannot be written, by the caller.

**SS$ IVLOGNAM**
The specified process name string has a length of 0 or has more than 15 characters.

**SS$ NONEXPR**
Warning. The specified process does not exist, or an invalid process identification was specified.

**SS$ NOPRIV**
The process does not have the privilege to wake the specified process.

Privilege Restrictions:

User privileges are required to wake:

- Other processes in the same group (GROUP privilege)
- Any other process in the system (WORLD privilege)
Notes:

1. If one or more wake requests are issued for a process that is not currently hibernating, a subsequent hibernate request completes immediately, that is, the process does not hibernate. No count is maintained of outstanding wakeup requests.

2. A hibernating process can also be awakened with the Schedule Wakeup ($SCHDWK) system service.

For an example of the $WAKE system service and a discussion of the hibernate/wake mechanism, see Section 3.5, "Process Control Services."
4.80 $WFLAND - WAIT FOR LOGICAL AND OF EVENT FLAGS

The Wait for Logical AND of Event Flags system service allows a process to specify a mask of event flags for which it wishes to wait. All of the indicated event flags within a specified event cluster must be set; otherwise, the process is placed in a wait state until they are all set.

Macro Format:

$WFLAND efn,mask

High-Level Language Format:

SYS$WFLAND(efn,mask)

efn
number of any event flag within the cluster being used.

mask
32-bit mask in which bits set to 1 indicate the event flags within the cluster that must be set.

Return Status:

SS$NORMAL
Service successfully completed.

SS$ILEFPC
An illegal event flag number was specified.

SS$UNASEFPC
The process is not associated with the cluster containing the specified event flag.

Note:

The wait state caused by this service can be interrupted by an asynchronous system trap (AST) if (1) the access mode at which the AST is to execute is less than or equal to the access mode from which the wait was issued and (2) the process is enabled for ASTs at that access mode.

When the AST service routine completes execution, the system repeats the $WFLAND request. If the specified event flags are all set, the process resumes execution.

For an example of the $WFLAND system service, see Section 3.1, "Event Flag Services."
$WFLOR

4.81  $WFLOR - WAIT FOR LOGICAL OR OF EVENT FLAGS

The Wait for Logical OR of Event Flags system service tests the event flags specified by a mask within a specified cluster and returns immediately if any of them is set. Otherwise, the process is placed in a wait state until at least one of the selected event flags is set.

Macro Format:

$WFLOR  efn ,mask

High-Level Language Format:

SYS$WFLOR(efn ,mask)

efn
  number of any event flag within the cluster being used.

mask
  32-bit mask in which bits set to 1 indicate the event flags of interest.

Return Status:

SS$NORMAL
  Service successfully completed.

SS$_ILLEFC
  An illegal event flag number was specified.

SS$_UNASEFC
  The process is not associated with the cluster containing the specified event flag.

Note:

The wait state caused by this service can be interrupted by an asynchronous system trap (AST) if (1) the access mode at which the AST is to execute is less than or equal to the access mode from which the wait was issued and (2) the process is enabled for ASTs at that access mode.

When the AST service routine completes execution, the system repeats the $WFLOR request. If any of the event flags has been set, the process resumes execution.
APPENDIX A

SYSTEM SYMBOLIC DEFINITION MACROS

This appendix summarizes system-provided macros that define symbolic values for use with system services, and lists the symbols defined by each macro. The macros listed in this appendix are:

<table>
<thead>
<tr>
<th>Macro</th>
<th>Symbols Defined</th>
</tr>
</thead>
<tbody>
<tr>
<td>$IODEF</td>
<td>Symbolic names for I/O function codes</td>
</tr>
<tr>
<td>$MSGDEF</td>
<td>Symbolic names to identify mailbox message senders</td>
</tr>
<tr>
<td>$PRDEF</td>
<td>Internal processor registers</td>
</tr>
<tr>
<td>$PRTDEF</td>
<td>Symbolic names for hardware protection codes</td>
</tr>
<tr>
<td>$PSLDEF</td>
<td>Processor status longword (PSL) mask and field definitions, and symbolic names for access modes</td>
</tr>
<tr>
<td>$SSDEKF</td>
<td>Symbolic names for system status codes</td>
</tr>
</tbody>
</table>

The symbolic definitions generated by each of these macros are listed on the following pages. Definitions generated by the following macros are listed elsewhere in this manual (consult the Index for page number references).

<table>
<thead>
<tr>
<th>Macro</th>
<th>Symbols Defined</th>
</tr>
</thead>
<tbody>
<tr>
<td>$ACCDEF</td>
<td>Accounting manager request type codes and process termination message and accounting record information offsets</td>
</tr>
<tr>
<td>$CHFDEF</td>
<td>Condition handler argument offsets</td>
</tr>
<tr>
<td>$DIBDEF</td>
<td>Device information buffer offsets</td>
</tr>
<tr>
<td>$JBCMSGDEF</td>
<td>Job controller return status codes</td>
</tr>
<tr>
<td>$JPIDEF</td>
<td>Job/process information request type codes</td>
</tr>
<tr>
<td>$OPCDEF</td>
<td>Operator communication manager request type codes, buffer offsets, and return status codes</td>
</tr>
<tr>
<td>$PQLDEF</td>
<td>Quota types for process creation quota list</td>
</tr>
<tr>
<td>$PRVDEF</td>
<td>User privileges</td>
</tr>
<tr>
<td>$SECDEF</td>
<td>Attribute flags for private/global section creation and mapping</td>
</tr>
<tr>
<td>$SMRDEF</td>
<td>Symbiont manager request type and option codes</td>
</tr>
</tbody>
</table>

A-1
SYSTEM SYMBOLIC DEFINITION MACROS

A.1 USING SYSTEM SYMBOLS

The default system macro library, STARLET.MLB, contains the macro definitions for system symbols. When you assemble a source program that calls any of these macros, the assembler automatically searches STARLET.MLB for the macro definitions.

Each symbol name has a unique numeric value. To obtain a list of symbols, in alphabetic order and in numeric order, use the following procedure:

1. Create a file with the file type of MAR containing the lines:

   $xxDEF   GLOBAL
   .END

   where xx is the prefix of the macro defining the symbols you need, for example $SSSDEF or $MSGDEF.

2. Assemble the file with the MACRO command:

   $ MACRO file-name

   where file-name is the file name of the file containing the $xxDEF macro call. The input file type defaults to MAR.

3. Link the object module created by the assembler, requesting the linker to create a full map file:

   $ LINK/MAP/FULL/NOEXE file-name

   The linker map file, named file-name.MAP, contains a list of all the symbols defined in the macro, in numeric order.

You can specify more than one macro in the same assembly source file to obtain the numeric values for more than one set of definitions.

A.2 $IODEF MACRO - SYMBOLIC NAMES FOR I/O FUNCTION CODES

The function codes and function modifiers defined in the $IODEF macro are grouped according to the devices for which the I/O operation is requested. For your convenience, the arguments (P1, P2, and so on), are also listed.
### A.2.1 Terminal Driver

<table>
<thead>
<tr>
<th>Functions</th>
<th>Arguments</th>
<th>Modifiers</th>
</tr>
</thead>
<tbody>
<tr>
<td>IO$_READVBLK</td>
<td>P1 - buffer address</td>
<td>IOSM_NOECHO</td>
</tr>
<tr>
<td>IO$_READLBLK</td>
<td>P2 - buffer size</td>
<td>IOSM_CVTLOW</td>
</tr>
<tr>
<td>IO$_READPBLK</td>
<td>P3 - timeout</td>
<td>IOSM_NOPILTR</td>
</tr>
<tr>
<td>IO$_READPROMPT</td>
<td>P4 - read terminator</td>
<td>IOSM_TIMED</td>
</tr>
<tr>
<td></td>
<td>block address</td>
<td>IOSM_PURGE</td>
</tr>
<tr>
<td></td>
<td>P5 - prompt string</td>
<td>IOSM_DSABLMBX</td>
</tr>
<tr>
<td></td>
<td>buffer address(^1)</td>
<td>IOSM_TRMNOECHO</td>
</tr>
<tr>
<td></td>
<td>P6 - prompt string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>buffer size(^1)</td>
<td></td>
</tr>
<tr>
<td>IO$_WRITEVBLK</td>
<td>P1 - buffer address</td>
<td>IOSM_CANCTRL0</td>
</tr>
<tr>
<td>IO$_WRITELBLK</td>
<td>P2 - buffer size</td>
<td>IOSM_ENABLMX</td>
</tr>
<tr>
<td>IO$_WRITEPBLK</td>
<td>P3 - (ignored)</td>
<td>IOSM_NOFORMAT</td>
</tr>
<tr>
<td></td>
<td>P4 - carriage control</td>
<td></td>
</tr>
<tr>
<td></td>
<td>specifier(^2)</td>
<td></td>
</tr>
<tr>
<td>IO$_SETMODE</td>
<td>P1 - characteristics</td>
<td></td>
</tr>
<tr>
<td>IO$_SETCHAR</td>
<td>buffer address</td>
<td></td>
</tr>
<tr>
<td></td>
<td>P2 - (ignored)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>P3 - speed specifier</td>
<td></td>
</tr>
<tr>
<td></td>
<td>P4 - fill specifier</td>
<td></td>
</tr>
<tr>
<td></td>
<td>P5 - parity flags</td>
<td></td>
</tr>
</tbody>
</table>

\(^1\) Only for IO$_READPROMPT

\(^2\) Only for IO$_WRITEPBLK and IO$_WRITEVBLK

### A.2.2 Disk Drivers

<table>
<thead>
<tr>
<th>Functions</th>
<th>Arguments</th>
<th>Modifiers</th>
</tr>
</thead>
<tbody>
<tr>
<td>IO$_READVBLK</td>
<td>P1 - buffer address</td>
<td>IOSM_DATACHECK</td>
</tr>
<tr>
<td>IO$_READLBLK</td>
<td>P2 - byte count</td>
<td>IOSM_INHRETRY</td>
</tr>
<tr>
<td>IO$_READPBLK</td>
<td>P3 - disk address</td>
<td>IOSM_INHSEEK(^1)</td>
</tr>
<tr>
<td>IO$_WRITEVBLK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO$_WRITELBLK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO$_WRITEPBLK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO$_SETMODE</td>
<td>P1 - characteristic buffer</td>
<td>IOSM_INHRETRY</td>
</tr>
<tr>
<td>IO$_SETCHAR</td>
<td>address</td>
<td></td>
</tr>
<tr>
<td>IO$_CREATE</td>
<td>P1 - FIB descriptor address</td>
<td>IOSM_CREATE(^2)</td>
</tr>
<tr>
<td>IO$_ACCESS</td>
<td>P2 - file name string address</td>
<td>IOSM_ACCESS(^2)</td>
</tr>
<tr>
<td>IO$_DEACCESS</td>
<td>P3 - result string length</td>
<td>IOSM_DELETE(^3)</td>
</tr>
<tr>
<td>IO$_MODIFY</td>
<td>P4 - result string descriptor address</td>
<td></td>
</tr>
<tr>
<td>IO$_DELETE</td>
<td>P5 - attribute list address</td>
<td></td>
</tr>
</tbody>
</table>

\(^1\) Only for IO$_READPBLK and IO$_WRITEPBLK

\(^2\) Only for IO$_CREATE and IO$_ACCESS

\(^3\) Only for IO$_CREATE and IO$_DELETE
### A.2.3 Magnetic Tape Drivers

<table>
<thead>
<tr>
<th>Functions</th>
<th>Arguments</th>
<th>Modifiers</th>
</tr>
</thead>
<tbody>
<tr>
<td>IOS$ READVBLK</td>
<td>P1 - buffer address</td>
<td>IO$M_DATACHECK</td>
</tr>
<tr>
<td>IOS$ READLBLK</td>
<td>P2 - byte count</td>
<td>IO$M_INHRETRY</td>
</tr>
<tr>
<td>IOS$ READPBLK</td>
<td></td>
<td>IO$M_REVERSE&lt;sup&gt;1&lt;/sup&gt;</td>
</tr>
<tr>
<td>IOS$ WRITEVBLK</td>
<td></td>
<td>IO$M_INHEXTGAP&lt;sup&gt;2&lt;/sup&gt;</td>
</tr>
<tr>
<td>IOS$ WRITELBLK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IOS$ WRITEPBLK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IOS$ SETMODE</td>
<td>P1 - characteristics buffer address</td>
<td>IO$M_INHRETRY</td>
</tr>
<tr>
<td>IOS$ SETCHAR</td>
<td></td>
<td>IO$M_INHEXTGAP&lt;sup&gt;2&lt;/sup&gt;</td>
</tr>
<tr>
<td>IOS$ CREATE</td>
<td>P1 - FIB descriptor address</td>
<td>IO$M_CREATE&lt;sup&gt;3&lt;/sup&gt;</td>
</tr>
<tr>
<td>IOS$ ACCESS</td>
<td>P2 - file name string</td>
<td>IO$M_ACCESS&lt;sup&gt;3&lt;/sup&gt;</td>
</tr>
<tr>
<td>IOS$ DEACCESS</td>
<td>address</td>
<td>IO$M_DMO&lt;/sup&gt;UNT&lt;sup&gt;4&lt;/sup&gt;</td>
</tr>
<tr>
<td>IOS$ MODIFY</td>
<td>P3 - result string length</td>
<td></td>
</tr>
<tr>
<td>IOS$ ACPCONTROL</td>
<td>address</td>
<td></td>
</tr>
<tr>
<td>IOS$ SKIPFILE</td>
<td>P1 - skip n tape marks</td>
<td>IO$M_INHRETRY</td>
</tr>
<tr>
<td>IOS$ SKIPRECORD</td>
<td>P1 - skip n records</td>
<td>IO$M_INHRETRY</td>
</tr>
<tr>
<td>IOS$ MOUNT</td>
<td>(none)</td>
<td></td>
</tr>
<tr>
<td>IOS$ REWIND</td>
<td>(none)</td>
<td>IO$M_INHRETRY</td>
</tr>
<tr>
<td>IOS$ REWINDOFF</td>
<td>(none)</td>
<td>IO$M_NOWAIT</td>
</tr>
<tr>
<td>IOS$ WRITEOF</td>
<td>(none)</td>
<td>IO$M_INHEXTGAP</td>
</tr>
<tr>
<td>IOS$ SENSEMODE</td>
<td>(none)</td>
<td>IO$M_INHRETRY</td>
</tr>
</tbody>
</table>

<sup>1</sup>Only for read functions
<sup>2</sup>Only for write functions
<sup>3</sup>Only for IOS$ CREATE and IOS$ ACCESS
<sup>4</sup>Only for IOS$ ACPCONTROL

### A.2.4 Line Printer Driver

<table>
<thead>
<tr>
<th>Functions</th>
<th>Arguments</th>
<th>Modifiers</th>
</tr>
</thead>
<tbody>
<tr>
<td>IOS$ WRITEVBLK</td>
<td>P1 - buffer address</td>
<td>(none)</td>
</tr>
<tr>
<td>IOS$ WRITELBLK</td>
<td>P2 - buffer size</td>
<td></td>
</tr>
<tr>
<td>IOS$ WRITEPBLK</td>
<td>P3 - (ignored)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>P4 - carriage control</td>
<td></td>
</tr>
<tr>
<td></td>
<td>specifier&lt;sup&gt;1&lt;/sup&gt;</td>
<td></td>
</tr>
<tr>
<td>IOS$ SETMODE</td>
<td>P1 - characteristics buffer address</td>
<td>(none)</td>
</tr>
<tr>
<td>IOS$ SETCHAR</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<sup>1</sup>Only for IOS$ WRITEVBLK and IOS$ WRITELBLK
### A.2.5 Card Reader Driver

<table>
<thead>
<tr>
<th>Functions</th>
<th>Arguments</th>
<th>Modifiers</th>
</tr>
</thead>
<tbody>
<tr>
<td>IO$ READLBLK</td>
<td>P1 - buffer address</td>
<td>IOSM_BINARY</td>
</tr>
<tr>
<td>IO$ READVBLK</td>
<td>P2 - byte count</td>
<td>IOSM_PACKED</td>
</tr>
<tr>
<td>IO$ READPBBLK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO$ SETMODE</td>
<td>P1 - characteristics</td>
<td>(none)</td>
</tr>
<tr>
<td>IO$ SETCHAR</td>
<td>buffer address</td>
<td></td>
</tr>
<tr>
<td>IO$ SENSEMODE</td>
<td>(none)</td>
<td></td>
</tr>
</tbody>
</table>

### A.2.6 Mailbox Driver

<table>
<thead>
<tr>
<th>Functions</th>
<th>Arguments</th>
<th>Modifiers</th>
</tr>
</thead>
<tbody>
<tr>
<td>IO$ READVBLK</td>
<td>P1 - buffer address</td>
<td>IOSM_NOW</td>
</tr>
<tr>
<td>IO$ READLBLK</td>
<td>P2 - buffer size</td>
<td></td>
</tr>
<tr>
<td>IO$ READPBBLK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO$ WRITEVBLK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO$ WRITELBLK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO$ WRITEPBBLK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO$ WRITEOF</td>
<td>(none)</td>
<td></td>
</tr>
<tr>
<td>IO$ SETMODE!IOSM_READATTN</td>
<td>P1 - AST address</td>
<td></td>
</tr>
<tr>
<td>IO$ SETMODE!IOSM_WRTATTN</td>
<td>P1 - AST parameter</td>
<td></td>
</tr>
</tbody>
</table>

### A.2.7 DMCLl Driver

<table>
<thead>
<tr>
<th>Functions</th>
<th>Arguments</th>
<th>Modifiers</th>
</tr>
</thead>
<tbody>
<tr>
<td>IO$ READLBLK</td>
<td>P1 - buffer address</td>
<td>IOSM_DSABLMBX(^1)</td>
</tr>
<tr>
<td>IO$ READPBBLK</td>
<td>P2 - message size</td>
<td>IOSM_NOW(^1)</td>
</tr>
<tr>
<td>IO$ READVBLK</td>
<td>P6 - diagnostic buffer(^2)</td>
<td>IOSM_ENABLMBX(^3)</td>
</tr>
<tr>
<td>IO$ WRITELBLK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO$ WRITEPBBLK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO$ WRITEVBLK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO$ SETMODE</td>
<td>P1 - characteristics</td>
<td></td>
</tr>
<tr>
<td>IO$ SETCHAR</td>
<td>buffer address</td>
<td></td>
</tr>
<tr>
<td>IO$ SETMODE!IOSM_ATTNAST</td>
<td>P1 - AST service routine address</td>
<td></td>
</tr>
<tr>
<td>IO$ SETCHAR!IOSM_ATTNAST</td>
<td>P2 - (ignored)</td>
<td></td>
</tr>
<tr>
<td>IO$ SETMODE!IOSM_SHUTDOWN</td>
<td>P1 - characteristics block address</td>
<td></td>
</tr>
<tr>
<td>IO$ SETCHAR!IOSM_SHUTDOWN</td>
<td>P2 - (ignored)</td>
<td></td>
</tr>
<tr>
<td>IO$ SETMODE!IOSM_STARTUP</td>
<td>P1 - characteristics block address</td>
<td></td>
</tr>
<tr>
<td>IO$ SETCHAR!IOSM_STARTUP</td>
<td>P2 - (ignored)</td>
<td></td>
</tr>
</tbody>
</table>

\(^1\) Only for IO$ READLBLK and IO$ READPBBLK  
\(^2\) Only for IO$ READPBBLK and IO$ WRITEPBBLK  
\(^3\) Only for IO$ WRITELBLK and IO$ WRITEPBBLK
A.2.8 ACP Interface Driver

<table>
<thead>
<tr>
<th>Functions</th>
<th>Arguments</th>
<th>Modifiers</th>
</tr>
</thead>
<tbody>
<tr>
<td>IOS$ CREATE</td>
<td>P1 - FIB descriptor address</td>
<td>IOSM CREATE(^1)</td>
</tr>
<tr>
<td>IOS$ ACCESS</td>
<td>P2 - file name string address</td>
<td>IOSM ACCESS(^1)</td>
</tr>
<tr>
<td>IOS$ DEACCESS</td>
<td>P3 - result string length address</td>
<td>IOSM DELETE(^2)</td>
</tr>
<tr>
<td>IOS$ MODIFY</td>
<td>P4 - result string descriptor address</td>
<td>IOSM DROUTE(^3)</td>
</tr>
<tr>
<td>IOS$ ACPCONTROL</td>
<td>P5 - attribute list address</td>
<td></td>
</tr>
<tr>
<td>IOS$ MOUNT</td>
<td>(none)</td>
<td></td>
</tr>
</tbody>
</table>

\(^1\)Only for IOS$ CREATE and IOS$ ACCESS
\(^2\)Only for IOS$ CREATE and IOS$ DELETE
\(^3\)Only for IOS$ ACPCONTROL

A.3 $MSGDEF MACRO - SYMBOLIC NAMES FOR SYSTEM MAILBOX MESSAGES

<table>
<thead>
<tr>
<th>Symbolic Name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSG$ TRMUNSOLIC</td>
<td>Unsolicited terminal data</td>
</tr>
<tr>
<td>MSG$ CRUNSOLIC</td>
<td>Unsolicited card reader data</td>
</tr>
<tr>
<td>MSG$ DELPROC</td>
<td>Delete process</td>
</tr>
<tr>
<td>MSG$ SNDMB</td>
<td>Send to symbiont manager</td>
</tr>
<tr>
<td>MSG$ DEVOPFLIN</td>
<td>Device offline</td>
</tr>
<tr>
<td>MSG$ TRMHANGUP</td>
<td>Terminal hangup</td>
</tr>
<tr>
<td>MSG$ DEVONLIN</td>
<td>Device online</td>
</tr>
<tr>
<td>MSG$ OPRQST</td>
<td>Operator request</td>
</tr>
<tr>
<td>MSG$ OPREPLY</td>
<td>Operator reply</td>
</tr>
<tr>
<td>MSG$ SMBINI</td>
<td>Symbiont is initiated</td>
</tr>
<tr>
<td>MSG$ SMBDON</td>
<td>Symbiont has finished</td>
</tr>
<tr>
<td>MSG$ SNDACC</td>
<td>Send to accounting manager</td>
</tr>
<tr>
<td>MSG$ XM_DATAVL</td>
<td>Data available (DMC-11)</td>
</tr>
<tr>
<td>MSG$ XM_SHUTDN</td>
<td>Unit shutdown (DMC-11)</td>
</tr>
<tr>
<td>MSG$ XM_ATTN</td>
<td>Unit attention (DMC-11)</td>
</tr>
<tr>
<td>MSG$ INITOPR</td>
<td>Initiate file printing</td>
</tr>
<tr>
<td>MSG$ ABOOPR</td>
<td>Abort printing a file</td>
</tr>
<tr>
<td>MSG$ SUSOPR</td>
<td>Pause printing a file</td>
</tr>
<tr>
<td>MSG$ RESOPR</td>
<td>Resume printing a file</td>
</tr>
<tr>
<td>MSG$ DELSMB</td>
<td>Symbiont should delete itself</td>
</tr>
<tr>
<td>MSG$ SMBRSP</td>
<td>Symbiont response</td>
</tr>
<tr>
<td>MSG$ ACCRSP</td>
<td>Accounting manager response</td>
</tr>
<tr>
<td>MSG$ ABORT</td>
<td>Network partner aborted link</td>
</tr>
<tr>
<td>MSG$ CONFIRM</td>
<td>Network connect confirm</td>
</tr>
<tr>
<td>MSG$ CONNECT</td>
<td>Network inbound connect initiate</td>
</tr>
<tr>
<td>MSG$ DISCON</td>
<td>Network partner disconnected-hangup</td>
</tr>
<tr>
<td>MSG$ EXIT</td>
<td>Network partner exited prematurely</td>
</tr>
<tr>
<td>MSG$ INTMSG</td>
<td>Network interrupt message; unsolicited data</td>
</tr>
<tr>
<td>MSG$ PATHLOST</td>
<td>Network path lost to partner</td>
</tr>
<tr>
<td>MSG$ PROTOCOL</td>
<td>Network protocol error</td>
</tr>
<tr>
<td>MSG$ REJECT</td>
<td>Network connect reject</td>
</tr>
<tr>
<td>MSG$ THIRDPARTY</td>
<td>Network third party disconnect</td>
</tr>
<tr>
<td>MSG$ TIMEOUT</td>
<td>Network connect timeout</td>
</tr>
</tbody>
</table>
## A.4 $PRDEF MACRO - SYMBOLIC NAMES FOR PROCESSOR REGISTERS

<table>
<thead>
<tr>
<th>Symbolic Name</th>
<th>Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>PR$ KSP</td>
<td>Kernel stack pointer</td>
</tr>
<tr>
<td>PR$ ESP</td>
<td>Executive stack pointer</td>
</tr>
<tr>
<td>PR$ SSP</td>
<td>Supervisor stack pointer</td>
</tr>
<tr>
<td>PR$ USP</td>
<td>User stack pointer</td>
</tr>
<tr>
<td>PR$ ISP</td>
<td>Interrupt stack pointer</td>
</tr>
<tr>
<td>PR$ P0BR</td>
<td>P0 base register</td>
</tr>
<tr>
<td>PR$ P0LR</td>
<td>P0 limit register</td>
</tr>
<tr>
<td>PR$ P1BR</td>
<td>P1 base register</td>
</tr>
<tr>
<td>PR$ P1LR</td>
<td>P1 limit register</td>
</tr>
<tr>
<td>PR$ SBR</td>
<td>System base register</td>
</tr>
<tr>
<td>PR$ SLR</td>
<td>System limit register</td>
</tr>
<tr>
<td>PR$ PCBB</td>
<td>Process control block base register</td>
</tr>
<tr>
<td>PR$ SCBB</td>
<td>System control block base register</td>
</tr>
<tr>
<td>PR$ IPL</td>
<td>Interrupt priority level register</td>
</tr>
<tr>
<td>PR$ ASTLVL</td>
<td>AST level register</td>
</tr>
<tr>
<td>PR$ SIRR</td>
<td>Software interrupt request register</td>
</tr>
<tr>
<td>PR$ SISR</td>
<td>Software interrupt summary register</td>
</tr>
<tr>
<td>PR$ MAPEN</td>
<td>Mapping enable register</td>
</tr>
<tr>
<td>PR$ TBI A</td>
<td>Translation buffer invalidate all</td>
</tr>
<tr>
<td>PR$ TBI S</td>
<td>Translation buffer invalidate single</td>
</tr>
<tr>
<td>PR$ ICCS</td>
<td>Interval clock control status register</td>
</tr>
<tr>
<td>PR$ NCR</td>
<td>Interval clock next interval register</td>
</tr>
<tr>
<td>PR$ ICR</td>
<td>Interval clock interval count register</td>
</tr>
<tr>
<td>PR$ TODR</td>
<td>Time of day register</td>
</tr>
<tr>
<td>PR$ RXCS</td>
<td>Console receiver control status register</td>
</tr>
<tr>
<td>PR$ RXDB</td>
<td>Console receiver data buffer register</td>
</tr>
<tr>
<td>PR$ TXCS</td>
<td>Console transmit control status register</td>
</tr>
<tr>
<td>PR$ TXDB</td>
<td>Console transmit data buffer register</td>
</tr>
<tr>
<td>PR$ ACCS</td>
<td>Accelerator control status register</td>
</tr>
<tr>
<td>PR$ ACCR</td>
<td>Accelerator reserved</td>
</tr>
<tr>
<td>PR$ WC SA</td>
<td>WCS address register</td>
</tr>
<tr>
<td>PR$ WC SD</td>
<td>WCS data register</td>
</tr>
<tr>
<td>PR$ SBI FS</td>
<td>SBI fault status register</td>
</tr>
<tr>
<td>PR$ SBI S</td>
<td>SBI silo register</td>
</tr>
<tr>
<td>PR$ SBI SC</td>
<td>SBI comparator register</td>
</tr>
<tr>
<td>PR$ SBI MT</td>
<td>SBI maintenance register</td>
</tr>
<tr>
<td>PR$ SBI ER</td>
<td>SBI error register</td>
</tr>
<tr>
<td>PR$ SBIT A</td>
<td>SBI timeout address register</td>
</tr>
<tr>
<td>PR$ SBI QC</td>
<td>SBI quadword clear register</td>
</tr>
<tr>
<td>PR$ SID</td>
<td>System identification register</td>
</tr>
</tbody>
</table>

## A.5 $PRTFDEF - HARDWARE PROTECTION CODE DEFINITIONS

<table>
<thead>
<tr>
<th>Symbolic Name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRT$C NA</td>
<td>No access</td>
</tr>
<tr>
<td>PRT$C KR</td>
<td>Kernel read only</td>
</tr>
<tr>
<td>PRT$C KW</td>
<td>Kernel write</td>
</tr>
<tr>
<td>PRT$C ER</td>
<td>Executive read only</td>
</tr>
<tr>
<td>PRT$C EW</td>
<td>Executive write</td>
</tr>
<tr>
<td>PRT$C SR</td>
<td>Supervisor read only</td>
</tr>
<tr>
<td>PRT$C SW</td>
<td>Supervisor write</td>
</tr>
<tr>
<td>PRT$C UR</td>
<td>User read only</td>
</tr>
<tr>
<td>PRT$C UW</td>
<td>User write</td>
</tr>
<tr>
<td>PRT$C ERKW</td>
<td>Executive read; kernel write</td>
</tr>
<tr>
<td>PRT$C SRKW</td>
<td>Supervisor read; kernel write</td>
</tr>
<tr>
<td>PRT$C SREW</td>
<td>Supervisor read; executive write</td>
</tr>
<tr>
<td>PRT$C URKW</td>
<td>User read; kernel write</td>
</tr>
<tr>
<td>PRT$C UREW</td>
<td>User read; executive write</td>
</tr>
<tr>
<td>PRT$C UR SW</td>
<td>User read; supervisor write</td>
</tr>
</tbody>
</table>
A.6  $PSLDEF MACRO - PROCESSOR STATUS LONGWORD SYMBOL DEFINITIONS

<table>
<thead>
<tr>
<th>Symbolic Name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSL$V_TBIT</td>
<td>TBIT enable field</td>
</tr>
<tr>
<td>PSL$S_TBIT</td>
<td>Length of TBIT enable field</td>
</tr>
<tr>
<td>PSL$M_TBIT</td>
<td>Mask for TBIT enable field</td>
</tr>
<tr>
<td>PSL$V_IV</td>
<td>Integer overflow field</td>
</tr>
<tr>
<td>PSL$S_IV</td>
<td>Length of integer overflow field</td>
</tr>
<tr>
<td>PSL$M_IV</td>
<td>Mask for integer overflow field</td>
</tr>
<tr>
<td>PSL$V_FU</td>
<td>Floating undefined field</td>
</tr>
<tr>
<td>PSL$S_FU</td>
<td>Length of floating undefined field</td>
</tr>
<tr>
<td>PSL$M_FU</td>
<td>Mask for floating undefined field</td>
</tr>
<tr>
<td>PSL$V_DV</td>
<td>Divide by zero field</td>
</tr>
<tr>
<td>PSL$S_DV</td>
<td>Length of divide by zero field</td>
</tr>
<tr>
<td>PSL$M_DV</td>
<td>Mask for divide by zero field</td>
</tr>
<tr>
<td>PSL$V_IPL</td>
<td>Interrupt priority field</td>
</tr>
<tr>
<td>PSL$S_IPL</td>
<td>Length of interrupt priority field</td>
</tr>
<tr>
<td>PSL$V_PRV</td>
<td>Previous processor mode field</td>
</tr>
<tr>
<td>PSL$S_PRV</td>
<td>Length of previous processor mode field</td>
</tr>
<tr>
<td>PSL$V_CUR</td>
<td>Current processor mode field</td>
</tr>
<tr>
<td>PSL$S_CUR</td>
<td>Length of current processor mode field</td>
</tr>
<tr>
<td>PSL$V_IS</td>
<td>Interrupt stack field</td>
</tr>
<tr>
<td>PSL$S_IS</td>
<td>Length of interrupt stack field</td>
</tr>
<tr>
<td>PSL$M_IS</td>
<td>Mask for interrupt stack field</td>
</tr>
<tr>
<td>PSL$V_FPD</td>
<td>First part done field</td>
</tr>
<tr>
<td>PSL$S_FPD</td>
<td>Length of first part done field</td>
</tr>
<tr>
<td>PSL$M_FPD</td>
<td>Mask for first part done field</td>
</tr>
<tr>
<td>PSL$V_TR</td>
<td>Trace trap pending field</td>
</tr>
<tr>
<td>PSL$S_TR</td>
<td>Length of trace trap pending field</td>
</tr>
<tr>
<td>PSL$M_TR</td>
<td>Mask for trace trap pending field</td>
</tr>
<tr>
<td>PSL$V_CM</td>
<td>Compatibility mode field</td>
</tr>
<tr>
<td>PSL$S_CM</td>
<td>Length of compatibility mode field</td>
</tr>
<tr>
<td>PSL$M_CM</td>
<td>Mask for compatibility mode field</td>
</tr>
</tbody>
</table>

Symbolic Names for Access Modes

<table>
<thead>
<tr>
<th>Symbolic Name</th>
<th>Access Mode</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSL$C_KERNEL</td>
<td>Kernel</td>
<td>0</td>
</tr>
<tr>
<td>PSL$C_EXEC</td>
<td>Executive</td>
<td>1</td>
</tr>
<tr>
<td>PSL$C_SUPER</td>
<td>Supervisor</td>
<td>2</td>
</tr>
<tr>
<td>PSL$C_USER</td>
<td>User</td>
<td>3</td>
</tr>
</tbody>
</table>

A.7  $SSSDEF MACRO - SYMBOLIC NAMES FOR SYSTEM STATUS CODES

The $SSSDEF macro instruction defines symbolic names for system service return status codes and for exception condition names. The "Type" column, below, indicates one of the following:

<table>
<thead>
<tr>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>Successful completion</td>
</tr>
<tr>
<td>Warning</td>
<td>Warning return</td>
</tr>
<tr>
<td>Error</td>
<td>Error return</td>
</tr>
<tr>
<td>Severe</td>
<td>Severe error return</td>
</tr>
<tr>
<td>Condition</td>
<td>Exception condition</td>
</tr>
<tr>
<td>Status Code</td>
<td>Type</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------</td>
</tr>
<tr>
<td>SS$ ABORT</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$ ACCONFICT</td>
<td>Warning</td>
</tr>
<tr>
<td>SS$ ACCVIO</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$ ACCVIO</td>
<td>Condition</td>
</tr>
<tr>
<td>SS$ ACPVAFUL</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$ ARTRES</td>
<td>Condition</td>
</tr>
<tr>
<td>SS$ ASTFLT</td>
<td>Condition</td>
</tr>
<tr>
<td>SS$ BADATTRIB</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$ BADCHECKSUM</td>
<td>Warning</td>
</tr>
<tr>
<td>SS$ BADADESCAPE</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$ BADFILEHDR</td>
<td>Warning</td>
</tr>
<tr>
<td>SS$ BADFILENAME</td>
<td>Warning</td>
</tr>
<tr>
<td>SS$ BADFILEVER</td>
<td>Warning</td>
</tr>
<tr>
<td>SS$ BADIMGHDR</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$ BADDIRECTORY</td>
<td>Warning</td>
</tr>
<tr>
<td>SS$ BADPARAM</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$ BADSTACK</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$ BEGOFFILE</td>
<td>Warning</td>
</tr>
<tr>
<td>SS$ BLOCKCNTERR</td>
<td>Warning</td>
</tr>
<tr>
<td>SS$ BREAK</td>
<td>Condition</td>
</tr>
<tr>
<td>SS$ BUFBYTALI</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$ BUFFEROVF</td>
<td>Success</td>
</tr>
<tr>
<td>SS$ BUGCHECK</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$ CANCEL</td>
<td>Warning</td>
</tr>
<tr>
<td>SS$ CHANINTLK</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$ CLIPRCXT</td>
<td>Warning</td>
</tr>
<tr>
<td>SS$ CMODSUPR</td>
<td>Condition</td>
</tr>
<tr>
<td>SS$ CMODUSER</td>
<td>Condition</td>
</tr>
<tr>
<td>SS$ COMPAT</td>
<td>Condition</td>
</tr>
<tr>
<td>SS$ CONTINUE</td>
<td>Success</td>
</tr>
<tr>
<td>SS$ CONTROLC</td>
<td>Success</td>
</tr>
<tr>
<td>SS$ CONTROLO</td>
<td>Success</td>
</tr>
<tr>
<td>SS$ CREATED</td>
<td>Success</td>
</tr>
<tr>
<td>SS$ CTRLERR</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$ DATACHECK</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$ DATAOVERUN</td>
<td>Warning</td>
</tr>
<tr>
<td>SS$ DEBUG</td>
<td>Condition</td>
</tr>
<tr>
<td>SS$ DECOVF</td>
<td>Condition</td>
</tr>
<tr>
<td>SS$ DEACTIVE</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$ DEVALLOC</td>
<td>Warning</td>
</tr>
<tr>
<td>SS$ DEVALRALLOC</td>
<td>Success</td>
</tr>
<tr>
<td>SS$ DEVAASSIGN</td>
<td>Warning</td>
</tr>
<tr>
<td>SS$ DEVFOREIGN</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$ DEVICEFULL</td>
<td>Warning</td>
</tr>
<tr>
<td>SS$ DEVMOUNT</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$ DEVNOTALLOC</td>
<td>Warning</td>
</tr>
<tr>
<td>SS$ DEVNOTMBX</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$ DEVNOTMOUNT</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$ DEVOFFLINE</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$ DIRFULL</td>
<td>Warning</td>
</tr>
<tr>
<td>SS$ DRIVER</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$ DUPFilename</td>
<td>Warning</td>
</tr>
<tr>
<td>SS$ DUPLNAM</td>
<td>Severe</td>
</tr>
<tr>
<td>Status Code</td>
<td>Type</td>
</tr>
<tr>
<td>------------</td>
<td>------------</td>
</tr>
<tr>
<td>SS$ ENDOFFILE</td>
<td>Warning</td>
</tr>
<tr>
<td>SS$ ENDOFUSRLBL</td>
<td>Warning</td>
</tr>
<tr>
<td>SS$ EXQUOTA</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$ FCPREADERR</td>
<td>Warning</td>
</tr>
<tr>
<td>SS$ FCPREPSRN</td>
<td>Warning</td>
</tr>
<tr>
<td>SS$ FCPREWNDERR</td>
<td>Warning</td>
</tr>
<tr>
<td>SS$ FCPSPPACERR</td>
<td>Warning</td>
</tr>
<tr>
<td>SS$ FCPWRTERR</td>
<td>Warning</td>
</tr>
<tr>
<td>SS$ FILACERR</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$ FILALRACC</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$ FILELOCKED</td>
<td>Warning</td>
</tr>
<tr>
<td>SS$ FILENUMCHK</td>
<td>Warning</td>
</tr>
<tr>
<td>SS$ FILESEQCHK</td>
<td>Warning</td>
</tr>
<tr>
<td>SS$ FILESTRUCT</td>
<td>Warning</td>
</tr>
<tr>
<td>SS$ FILNOTACC</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$ FILNOTCNTG</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$ FILNOTEXP</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$ FLTDIV</td>
<td>Condition</td>
</tr>
<tr>
<td>SS$ FLTTOF</td>
<td>Condition</td>
</tr>
<tr>
<td>SS$ FLTUND</td>
<td>Condition</td>
</tr>
<tr>
<td>SS$ FORMAT</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$ GPTFULL</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$ GSDFULL</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$ HANGUP</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$ HEADERFULL</td>
<td>Warning</td>
</tr>
<tr>
<td>SS$ IDXFILEFULL</td>
<td>Warning</td>
</tr>
<tr>
<td>SS$ ILLBLKNUM</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$ ILLCNRTRFUNC</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$ ILLSEQC</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$ ILLIOPUNC</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$ ILLBLAST</td>
<td>Warning</td>
</tr>
<tr>
<td>SS$ ILLPAGCNT</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$ ILLSEQOP</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$ ILLSEQOP</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$ ILLUSRLBLRD</td>
<td>Warning</td>
</tr>
<tr>
<td>SS$ ILLUSRLBLWT</td>
<td>Warning</td>
</tr>
<tr>
<td>SS$ INCVOLLABEL</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$ INSFARG</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$ INSFMEM</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$ INSFRAME</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$ INFSWSL</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$ INTDIV</td>
<td>Condition</td>
</tr>
<tr>
<td>SS$ INTOVF</td>
<td>Condition</td>
</tr>
<tr>
<td>SS$ IVADDERR</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$ IVCHAN</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$ IVCHNLSEC</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$ IVDEVNAM</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$ IVGSDNAM</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$ IVLOGNAM</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$ IVLOGTAB</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$ IVPROMPT</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$ IVQUOTAL</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$ IVSECFLG</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$ IVSECIDCTL</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$ IVSSQRT</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$ IVSTSPLG</td>
<td>Severe</td>
</tr>
<tr>
<td>Status Code</td>
<td>Type</td>
</tr>
<tr>
<td>----------------------</td>
<td>--------</td>
</tr>
<tr>
<td>SS$ IVTIME</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$_LCKPAGFUL</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$ LENVIO</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$_LKWSETFUL</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$_MBFULL</td>
<td>Warning</td>
</tr>
<tr>
<td>SS$_MBTOOSML</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$_MCHECK</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$_MEDOFL</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$_MSGNOTFOUND</td>
<td>Success</td>
</tr>
<tr>
<td>SS$_MTLBLONG</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$_MUSTCLOSEPL</td>
<td>Warning</td>
</tr>
<tr>
<td>SS$_NOAQB</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$_NODATA</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$_NOHANDLER</td>
<td>Warning</td>
</tr>
<tr>
<td>SS$_NOHOMEBLK</td>
<td>Warning</td>
</tr>
<tr>
<td>SS$_NOIOCHAN</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$_NOLINKS</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$_NOLOGNAM</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$_NOMBX</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$_NOMOREFILES</td>
<td>Warning</td>
</tr>
<tr>
<td>SS$_NONEXDRV</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$_NONEXPR</td>
<td>Warning</td>
</tr>
<tr>
<td>SS$_NONLOCAL</td>
<td>Warning</td>
</tr>
<tr>
<td>SS$_NOPRIV</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$_NORMAL</td>
<td>Success</td>
</tr>
<tr>
<td>SS$_NOSIGNAL</td>
<td>Warning</td>
</tr>
<tr>
<td>SS$_NOSOLICIT</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$_NOSUCHDEV</td>
<td>Warning</td>
</tr>
<tr>
<td>SS$_NOSUCHFILE</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$_NOSUCHNODE</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$_NOSUCHSEC</td>
<td>Warning</td>
</tr>
<tr>
<td>SS$_NOTAPEOP</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$_NOTFILEDEV</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$_NORTINTBLSZ</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$_NORLABELMT</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$_NOTMODIFIED</td>
<td>Success</td>
</tr>
<tr>
<td>SS$_NOTNETDEV</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$_NOTTRAN</td>
<td>Success</td>
</tr>
<tr>
<td>SS$_NOTSQDEV</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$_OPCCUS</td>
<td>Condition</td>
</tr>
<tr>
<td>SS$_OPCDEC</td>
<td>Condition</td>
</tr>
<tr>
<td>SS$_OPINCOMPL</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$_PAGOWNVIO</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$_PAGRDERR</td>
<td>Condition</td>
</tr>
<tr>
<td>SS$_PARITY</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$_PARTEscape</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$_PFMBSY</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$_RADRMODE</td>
<td>Condition</td>
</tr>
<tr>
<td>SS$_REJECT</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$_REMOTE</td>
<td>Success</td>
</tr>
<tr>
<td>SS$_RESIGNAL</td>
<td>Warning</td>
</tr>
<tr>
<td>SS$_RESULTOVF</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$_ROPRAND</td>
<td>Condition</td>
</tr>
<tr>
<td>SS$_SECTBFLP</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$_SSFFAIL</td>
<td>Condition</td>
</tr>
<tr>
<td>SS$_SUBRNG</td>
<td>Condition</td>
</tr>
<tr>
<td>Status Code</td>
<td>Type</td>
</tr>
<tr>
<td>---------------------</td>
<td>----------</td>
</tr>
<tr>
<td>SS$ SUPERSEDE</td>
<td>Success</td>
</tr>
<tr>
<td>SS$ TAPEPOSLOST</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$ TBIT</td>
<td>Condition</td>
</tr>
<tr>
<td>SS$ TIMEOUT</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$ TOOMANYVER</td>
<td>Warning</td>
</tr>
<tr>
<td>SS$ TOOMUCHDATA</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$ UNASEPC</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$ UNSAFE</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$ UNWIND</td>
<td>Warning</td>
</tr>
<tr>
<td>SS$ UNWINDING</td>
<td>Warning</td>
</tr>
<tr>
<td>SS$ VASFULL</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$ VECINUSE</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$ VOLLINV</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$ WAITUSRLBL</td>
<td>Warning</td>
</tr>
<tr>
<td>SS$ WASCLR</td>
<td>Success</td>
</tr>
<tr>
<td>SS$ WASECC</td>
<td>Success</td>
</tr>
<tr>
<td>SS$ WASSET</td>
<td>Success</td>
</tr>
<tr>
<td>SS$ WRITLCK</td>
<td>Severe</td>
</tr>
<tr>
<td>SS$ WRONGACP</td>
<td>Severe</td>
</tr>
</tbody>
</table>
APPENDIX B

PROGRAM EXAMPLES

The sample programs presented on the following pages are self-documenting. Note that these programs do not perform any useful work; they are intended only to illustrate how to call various system services.

.TITLE ORION SYSTEM SERVICES TEST
.IDENT /01/

; ORION uses the following system services:

; .ASSIGN (Assign I/O Channel)
; .OUTPUT (form of Queue I/O Request and Wait For Event Flag)
; .NUMTIM (Convert Binary Time to Numeric Time)
; .BINTIM (Convert ASCII String to Binary Time)
; .SETIMR (Set Timer)
; .WAITFR (Wait for Single Event Flag)
; .READEF (Read Event Flags)
; .SETPRN (Set Process Name)

; This sample program illustrates:

; 1. Assigning an I/O channel to a terminal and writing messages
; to the terminal. The device name is specified by the logical name
; terminal. Before ORION is run, the logical name must be assigned
; an equivalence device name.

; 2. Using the .NUMTIM system service to find out whether the
; current time is before or after noon. A call to .SETIMR is
; made conditionally if the time is prior to noon.

; 3. How to obtain a delta time value in the system format to use
; as input to the Set Timer (.SETIMR) system service.

; 4. Calls to the Set Timer system service.

; A. Event flag - The .SETIMR call is followed by a wait for the
; specified event flag. When the timer expires, the program calls
; .READEF and displays the current status of the event flag
; cluster.

; B. ASR routine - one ASR routine is for a delta time interval.
; The other (conditional) is for an absolute time. In either
; case, the program continues execution and will be interrupted
; when the timer requests are processed.

; 5. An example of terminal input. The program prompts for a character
; string to be used as the process name of the current process.
; Then it uses this name as input to the .SETPRN system service.

B-1
.PAGE
.SBITL SYMBOLS AND DATA AREAS

; Macro library calls

$IFDEF
$SSIDF
$READEF

; Define I/O function codes
; Define system status values
; Define offsets for $READEF

; Local macros defined in private macro library

; DESCRIPTOR
; Generate character string descriptors

.MACRO DESCRIPTOR TEXT,?LABEL1,?LABEL2
.LONG LABEL2-LABEL1
.LONG LABEL1
LABEL1: .ASCII "TEXT"
LABEL2: .ENDM DESCRIPTOR

; MESSAGE
; Output messages formatted by FAO

.MACRO MESSAGE
$OUTPUT CHAN=TTCHAN,BUFFER=FAOBUF,LENGTH=FAOLEN
BSBW ERROR
.ENDM MESSAGE

.PSELECT RODATA,NOWRT,NOEXEC
.SBITL Read-Only Data Areas

; Local Read/Write Data

; LIST MEB

TTNAME: DESCRIPTOR <TERMINAL> ; Terminal logical name

; FAO control strings and data for timer (AST and event flag) tests

ASCNNO: DESCRIPTOR <-- 12:00:00.00> ; Noon in ASCII format
TENSEC: DESCRIPTOR <0 00:00:10> ; Ten seconds delta time in ASCII format
DISPLAYFN:
; Display cluster contents
; DESCRIPTOR <CLUSTER 2 CONTENTS: !XL>
TIMSTR:
; Display message after event flag wait
; DESCRIPTOR <!/TIMER ENTRY PROCESSED; CLUSTER 2 = !XL>
NODNMSEG:
; ASCII /I'M YOUR TIME AST ROUTINE; IT'S NOON.../
SECMSGSDESC:
; Display message from AST routine
; DESCRIPTOR <!TIME AST ROUTINE; DELTA TIME !XT>
TWENTY: .LONG -10*1000*1000*20,-1 ; 20 seconds delta time

B-2
.PAGE
;
; Announcement messages
;
FAOSTR:
   DESCRIPTOR <!/ORION: !AC >
   ; Master control string
   ; Name, message
;
; Announcement messages and lengths for outputting
;
HELLO:  .ASCII    /HELLO...MY NAME IS ORION.../
HELLOLEN:  .LONG HELLOLEN-HELLO
TIMERMSG:  .ASCII    /BEGINNING TIMER TESTS.../
TIMERLEN:  .LONG TIMERLEN-TIMERMSG
EFNWAITMSG:  .ASCII    /TIMER SET; WAIT TEN SECONDS/
EFNWAITLEN:  .LONG EFNWAITLEN-EFNWAITMSG
ASTWAITMSG:  .ASCII    /TIMER SET; AST IN 20 SECONDS/
ASTWAITLEN:  .LONG ASTWAITLEN-ASTWAITMSG
;
; Prompt for terminal input
PROMPT:  .ASCII    /ENTER 1-15 CHARACTER NAME FOR PROCESS:
PROMPTLEN:  .LONG PROMPTLEN-PROMPT
;
; Error message control strings
;
ERRSTR   DESCRIPTOR <!/SYSTEM SERVICE ERROR AT APP. !XL R0=!XL>
IDERRSTR   DESCRIPTOR <!/I/O ERROR; I0SB !XW>
BADASTSTR   DESCRIPTOR <BAD AST PARAMETER !UL>
WAKEUP:  .ASCII    /AWAKENED.../
WAKEUPLEN:  .LONG WAKEUPLEN-WAKEUP

.PAGE
.PSEC   RWDATA,RO,RWT,NOEXE
.SBTTL  Read and Write Data Areas
;
; Read/write data
;
;
; FAO control strings and buffer for all announcement messages
;
B-3
PROGRAM EXAMPLES

FAODESC:
.LONG 80 ;Descriptor for FAO output buffer
.LONG FAOBUF ;Address of buffer
FAOBUF: .BLKB 80 ;FAO buffer
FAOLEN: .WORD 0 ;Length of final string; always
.WORD 0 ;Need longword for $OUTPUT

; Buffer to format messages from AST routine; a separate output buffer
; ensures that if the AST is delivered while another message is being
; written into the FAO output buffer, no data or message will be lost.
;
FASTDESC:
.LONG 80 ;Descriptor for FAO output buffer
.LONG FASTBUF
FASTBUF: .BLKB 80 ;FAO buffer
FASTLEN: .WORD 0 ;Length of final string; always
.WORD 0 ;Need longword for $OUTPUT

; Receive channel number assigned to terminal and I/O status here
;
TTCHAN: .BLK 1 ;Terminal channel
TTOISB:
.BLK 1 ;$IOSB for terminal input
TTLLEN: 1 ;Return status
.BLK 1 ;Length of I/O
.BLK 1 ;Device char

; Argument list for $NAME.G form of a system service call
;
READLIST:
$READEF EFN=32;STATE=EFNTEST

; Buffer to obtain numeric values of components of time. Since
; the only field of interest is the hours field, the remaining
; fields in the buffer are not formatted.
;
TIMES: .BLK 3 ;Year, month, day
HOURS: 3 ;Current time in hours
.BLK 1 ;Remainder of buffer

; Buffer for terminal input (will create input descriptor for
; $SETPRN system service)

NAMEDESC:
.LONG 15 ;Descriptor setup
.LONG NAME ;Initial size of buffer
.NAME ;Address of buffer

;Fields for timer tests
;
NOON: .BLKB 1 ;Will contain 12:00 noon in system format
TEN: 1 ;Will contain 10 second delta time
EFNTEST:
  .LONG  0  ;Receive status of event flags
EFNTEST2:
  .LONG  0  ;Status after timer test

; Longword to save PC on entry to error handling subroutine

SAVEPC:  .BLKL  1

.PAGE
.SETTL TEST TIMERS WITH EVENT FLAGS AND ASTS
.PSECT TIMER,exe,newr

ORION:
  .WORD "M<R2,R3,R4,R5,R6>"  ;Entry mask

; Assign an I/O channel to the device specified by the logical name
; TERMINAL and issue a message indicating we’re off and running.
; Do not perform normal error checking here; instead, let the
; command interpreter issue a message based on the status in R0
; if the channel assignment fails.

SETUP:
  $ASSIGN_S DEVNUM=TTNAME,CHAN=TTCHAN
  BLDS R0,10\$  ;All okay, continue
  RET  ;Otherwise exit with status in R0

10\$:  $OUTPUT CHAN=TTCHAN,BUFFER=HELLO,LENGTH=HELLOLEN
  BSBW ERROR

; Call Read Event Flags to set status of event flags before beginning
; tests and use FAO to output the contents of local event flag cluster 2

$REDEFINE G READLST
$FAO S CTRSTR=DISPLAYEFN,OUTBUF=FAODESC,OUTLEN=FAOLEN,-
  P1=EFNTEST
  MESSAGE

; Announce start of timer tests

TIMTEST:
  $OUTPUT CHAN=TTCHAN,BUFFER=TIMERMESG,LENGTH=TIMERLEN
  BSBW ERROR

; Call $NUMTMR to find out if it is currently AM or PM. If
; the program is being run in the AM (any time), we’ll call
; $SETIMR to notify us via an AST when the time rolls over
; to afternoon. If it’s already PM, skip this setting of
; the timer.

$NUMTMR S TIMBUF=TIMES
  BSBW ERROR
  CMPW HOURS,#12  ;Before or afternoon?
  BGEQ 10\$  ;After, skip setting timer

; Fall through here: format ASCII string representing 12 noon
; into system quadword time format and call $SETIMR with
; the address of AST service routine to handle timer requests.
PROGRAM EXAMPLES

$BINTIM_S TIMBUF=ASCNOON,TIMADR=NOON ;Get binary noon time
BSBW  ERROR ;Error check

$SETIMR_S DAYTIM=NOON,ASTADR=TIMEAST,REQIDT=$12
BSBW  ERROR ;Error check

; Now, set a delta time of 10 seconds formatted into a quadword

10$:  $BINTIM_S TIMBUF=TENSEC,TIMADR=TEN ;Get binary delta time
BSBW  ERROR ;Error check
$SETIMR_S EFN=$33,DAYTIM=TEN ;Set timer (ten seconds)
BSBW  ERROR ;Error check

; Announce wait for event flag and wait; then read the
; event flag cluster and output its contents

$OUTPUT  CHAN=TTCHAN,BUFFER=EFNWAITMSG,LENGTH=EFNWAITLEN
$WAITFR_S EFN=$33 ;Now wait
BSBW  ERROR ;Error check

; Update argument list for $READEF and then call it with new address
; to write the cluster into. When complete, format a message and
; display the contents of the cluster.

MOVAL  EFNTEST2,READLST,READEF*STATE
$READEF_S READLST
BSBW  ERROR ;Error check
$FAO_S CTRSTR=TIMSTR,OUTLEN=FAOLEN,OUTBUF=FAODESC,-
P1=EFNTEST2
BSBW  ERROR ;Error check
MESSAGE

; Announce setting of timer with AST in 20 seconds (using
; alternate method of coding delta time). Then, set timer
; and continue.

$OUTPUT  CHAN=TTCHAN,BUFFER=ASTWAITMSG,LENGTH=ASTWAITLEN
$SETIMR_S DAYTIM=TWENTY,ASTADR=TIMEAST,REQIDT=$20
BSBW  ERROR ;Error check

.PAGE
; Issue a prompt for terminal input: request a name for the current
; process and then use the character strings entered as the process
; name.

RDNAME:
$OUTPUT  CHAN=TTCHAN,BUFFER=PROMPT,LENGTH=PROMPTLEN
BSBW  ERROR ;Error check
$INPUT  CHAN=TTCHAN,BUFFER=NAME,LENGTH=NAMEDESC,-
IOSB=TTIOSB
BSBW  ERROR

B-6
PROGRAM EXAMPLES

CMFW TTIOSB,#SS+..NORMAL I/O successful?
BEQL 10$ !Yes, do on
$FA0.S CTRSTR=IQERRSTR,OUTLEN=FAOLEN,OUTBUF=FAODESC,- 
P1=TTIOSB
MESSAGE 
BRW RDNAME !Go try again
MOVZUL TTTLEN,NAMEDESC !Update descriptor length
$SETFRN..S PRCNAM=NAMEDESC !Set process name
BBW ERROR

; Hibernate. When ORION is run interactively, the terminal is dormant.
; When the AST for the Set Timer service is delivered, ORION
; will awaken long enough to execute the AST service routine and
; then resume execution.
;
; If ORION is run in a subprocess, wakeups can be scheduled for
; delta time intervals. Each time it is awakened, ORION displays a
; message and then resumes hibernating.
;
HIB: $HIBER..S !For now
$OUTPUT CHAN=TTCHAN,BUFFER=WAKEUP,LENGTH=WAKEUPLEN
BRB HIB
RET

.PAGE .SBTTL AST ROUTINE TO HANDLE TIMER ENTRIES
TIMEAST:

.WORD 0 !Entry mask for timer AST routine
CMPL $12,4(AP) !Is it noon AST?
BEQL 10$ !Yes, do do it
CMPL $20,4(AP) !Is it delta time AST?
BEQL 20$ !Yes, do do that
BRW 30$ !Neither, issue error message

; Format message for noon AST
10$: $FA0..S CTRSTR=FAOSTR,OUTBUF=FASTDESC,OUTLEN=FASTLEN,P1=#NOONMSG
BBW ERROR !Error check
$OUTPUT CHAN=TTCHAN,BUFFER=FASTBUF,LENGTH=FASTLEN
BBW ERROR !Error check
RET

; Format message for 20 second AST
20$: $FA0..S CTRSTR=SECMSGDESC,OUTBUF=FASTDESC,OUTLEN=FASTLEN, 
P1=#TWENTY
$OUTPUT CHAN=TTCHAN,BUFFER=FASTBUF,LENGTH=FASTLEN
RET

; Format message if spurious AST
30$: $FA0..S CTRSTR=BADASTSTR,OUTLEN=FASTLEN,OUTBUF=FASTDESC, 
P1=4(AP)
$OUTPUT CHAN=TTCHAN,BUFFER=FASTBUF,LENGTH=FASTLEN
RET

.PAGE .SBTTL ERROR HANDLING ROUTINE

; Error handling routine: checks status code in R0.
; If low bit set, returns to mainline routine. Otherwise,
; displays approximate PC and R0 when system service call
; encounters an error and issues RET that causes image exit.

B-7
PROGRAM EXAMPLES

ERROR:
BLBC R0,10$
RSP
; If error, branch
; Otherwise, continue

; Use FAO to format output error message
10%:
MOVLR (SP),SAVEPC
$FAO_S CTRSTR=ERRSTR,OUTLEN=FAOLEN,OUTBUF=FAODESC,-
P1=SAVEPC,P2=R0
BLBC R0,END
$OUTPUT CHAN=TTCHAN,BUFFER=FAobuf,LENGTH=FAOLEN

END:
RET
.END ORION
.`TITLE CYGNUS SYSTEM SERVICES TEST PROGRAM
.IDENT /01/

; CYGNUS shows examples of the following system services:
; $TRNLOG - Translate Logical Name
; $ASSIGN - Assign I/O Channel
; $DCLEXH - Declare Exit Handler
; $CREMBX - Create Mailbox
; $GETCHN - Get I/O Channel Device Information
; $CREPRC - Create Process
; $FAO - Formatted ASCII Output
; $QIO - Queue I/O Request
; $CRELOG - Create Logical Name
; $WAKE - Wake Process
; $SETSFM - Set System Service Failure Exception Mode
; $WAITFR - Wait for Single Event Flag
; $DELLLOG - Delete Logical Name
; $DASSGN - Deassign I/O Channel

; This sample program illustrates:
; 1. Assigning a channel to the current output device by translating
; the logical name SYS$OUTPUT.
; 2. Declaring an exit handler to receive control at image exit.
; The exit handler ensures that the image exits in a graceful
; manner.
; 3. Creating a mailbox and using the $GETCHN system service
; to obtain the unit number.
; 4. Creating a subprocess and using the mailbox created as a
; termination mailbox. When the subprocess terminates, an AST
; service routine interprets the message.
; 5. Placing names in the group logical name table.
; 6. Waking a hibernating subprocess. The subprocess created by this
; program places itself in hibernation after getting started.
; When awakened, it translates the logical names placed in the
; group logical name table.

.PAGE

; System macro definitions required by CYGNUS

$SSDEF     ; Define status codes for returns
$IQODEF    ; Define I/O functions codes for $QIO
$MSGDEF    ; Define names for mailbox messages
$PQLDEF    ; Define names for quota list
$ACCDDEF   ; Define names for termination message
$DIBDEF    ; Define names for device information buffer

; Local macros:
; DESCRIPTOR, to define input character string descriptors for
; system service calls

.MACRO DESCRIPTOR TEXT,?LABEL1,?LABEL2
   .LONG   LABEL2-LABEL1
   .LONG   LABEL1
LABEL1:  .ASCII   TEXT
LABEL2:  .TEXT
.ENDM   DESCRIPTOR

B-9
PROGRAM EXAMPLES

; MESSAGE to output messages formatted by FAO

.Macro MESSAGE
$OUTPUT CHAN=TTCHAN, BUFFER=FAOBUF, LENGTH=FAOLEN
BSBW ERROR
.Endm MESSAGE

; Local macro: GRPNAME, to place logical name/equivalence name pairs in the group logical name table with $CRELOG and do error checking

.Macro GRPNAME LOGICAL=EQUAL
$CRELOG..TABLE=1,LOGNAME=LOGICAL, EQLNAME=EQUAL BSBW ERROR
.Endm GRPNAME

; Read/only data areas

.Psect RODATA, NOWRT, NOEXE
.List MEB

; Descriptor for input logical name

OUTPUT: DESCRIPTOR <SYS$OUTPUT>

; Buffers for announcement messages and lengths

HELLO: DESCRIPTOR <CYGNUS...HELLO>
HELLOLEN: .Long HELLOLEN-HELLO

BYE: .ASCII /CYGNUS EXIT HANDLER.../
BYELEN: .Long BYELEN-BYE

; Control strings for output messages formatted by FAO and associated counted ASCII strings to insert in messages

PRCSTR: DESCRIPTOR <LYRA CREATED, PID !XL> ;display PID of subprocess

ASTERRSTR: DESCRIPTOR <!MAILBOX MESSAGE HAS !AC !XW>

IDERR: .ASCII 'I/O ERROR' ;I/O error in AST routine
IDERR: .ASCII /BAD MSG ID/ ;Mailbox message not termination message

PIDERRSTR: DESCRIPTOR <!/SPURIOUS PROCESS ID !XL IN DELETION MAILBOX>

DONESTR: DESCRIPTOR <!/LYRA COMPLETED; STATUS !XL TIME !T>

BADEXSTR: DESCRIPTOR <!/EXIT DUE TO ERROR !XL>

; Descriptor to define name of image for subprocess to execute.


B-10
PROGRAM EXAMPLES

LYRAEXE:

; DESCRIPTOR <LYRA.EXE>

; Quota list for sub-process; defines minimal quotas required for
; for the sub-process to execute and ensures that the creating
; image will have sufficient quotas to continue.

QLIST: .BYTE POL%_BYTLIM ; Buffer quota
; .LONG 1024
; .BYTE POL%_FILMLIM ; Open file quota
; .LONG 3
; .BYTE POL%_PGFLQUOTA ; Paging file quota
; .LONG 256
; .BYTE POL%_PRCLLM ; Sub-process quota
; .LONG 1
; .BYTE POL%_TQUELM ; Timer queue quota
; .LONG 3
; .BYTE POL%_LISTEND

; Logical name/equivalence name pairs for group table.
; Note that one of the names is recursive in the table.

ORION: DESCRIPTOR <ORION>
HUNTER: DESCRIPTOR <HUNTER>
PEGASUS: DESCRIPTOR <PEGASUS>
HORSE: DESCRIPTOR <HORSE>
LYRA: DESCRIPTOR <LYRA>
HARP: DESCRIPTOR <HARP>
CYG: DESCRIPTOR <CYGNUS>
SWAN: DESCRIPTOR <SWAN>
DUCK: DESCRIPTOR <UGLY DUCKLING>
TALE: DESCRIPTOR <FAIRY TALE>

.PAGE

; Read/write data areas

.PSECT RWDATA,RD,WRT,NOEXE

TICHAN: .BLKW 1 ; Channel number of terminal

; Output buffer to receive physical terminal name

TTNAME: .LONG 63 ; Descriptor length
TTADDR: .LONG TT ; Address of buffer
TT: .BLKB 63 ; Maximum logical name length

; Termination control block

EXITBLOCK: .BLKL 1 ; Exit control block
; .LONG EXITRTN ; System uses this for pointer
; .LONG 2 ; Address of routine
; .LONG STATUS ; Number of arguments for handler
; .LONG STATUS ; Address to store status
; .BLKL 1 ; Store PC (if error)
; .BLKL 1 ; Status code at exit

; Fields used for termination mailbox creation, message buffering

B-11
PROGRAM EXAMPLES

EXCHAN: .BLW 1
EXITBUF:
.LONG ENDBUF-BBUF
.LONG BBUF
BBUF: .BLKL DIMK_LENGTH
ENDBUF:
MBXIOSB:
.BLW 1
MBLEN: .BLW 1
MBPID: .BLKL 1
EXITMSG:
.BLKL ACCK_TERMLEN

; ; Receive PID of subprocess here
;
LYRAPID:
.BLKL 1

.PAGE

; Output buffers for strings formatted by FAO
;
FAODESC: .LONG 80
.LONG FAOBUF
; Descriptor for output buffer
; 80-character buffer
; Address
FAOBUF: .BLKB 80
; Buffer
FAOLEN: .BLW 1
.BLW 1
; Receive length here
; Need longword for $QIO

; Need separate FAO buffers for use in AST routine to ensure
; that data doesn’t get clobbered asynchronously

FASTDESC: .LONG 80
.LONG FASTBUF
; Length
; Address
FASTBUF: .BLKB 80
; Buffer
FASTLEN: .BLW 1
.BLW 1
; Get length
; Need longword for $QIO

.PAGE
.PSECT CODE,EXE, RD, NOWRT
CYGNUS: .WORD 0
; Entry mask

; First, translate logical name SYS$OUTPUT to find name of
; current output device. If the image is run interactively,
; its equivalence name is system-defined, and will contain
; a 4-byte header. The program must check for the header and update
; the descriptor so the device name will be valid for calling $ASSIGN.
;
$TRNLLOG_S LOGNAME=OUTPUT, RSLLEN=TTNAME, RSLBUF=TTNAME
BSBW ERROR
CMPB TT, $"X1B
BNEQ 10$
SUBL $4, TTNAME
$Subtract 4 from length of name
ADDL $4, TTADDR
$Add 4 to address in descriptor

; Call $ASSIGN to assign an I/O channel and issue message verifying
; successful initialization

B-12
PROGRAM EXAMPLES

10$: $ASSIGN_S DEVNAM=TTNAME,CHAN=TTCHAN
     BSBW   ERROR ; Error check
     $OUTPUT CHAN=TTCHAN,BUFFER=HELLO,LENGTH=HELLOLEN
     BSBW   ERROR

; Declare exit handler to do cleanup operations

; $DCLEXH_S DESBLK=EXITBLOCK
     BSBW   ERROR

; Create a mailbox for sub-process termination message, then
; set the unit number of the mailbox by doing a $GETCHN

MAILBOX:
     $CREMBX_S CHAN=EXCHAN,MAXMSG=120,BUFQUO=240,PROMSK=0
     BSBW   ERROR
     $GETCHN_S CHAN=EXCHAN,PRIBUF=EXITBUF
     BSBW   ERROR

; Create the sub-process. Since the logical name SYS$OUTPUT
; has already been translated, the same equivalence name can be
; given to LYRA as its logical output device.
; LYRA will be able to assign a channel to this device as well.
; The MBXUNT argument specifies the name of the mailbox just
; created; the mailbox will receive a message when LYRA exits.

PROCESS:
     $CREPRC_S IMAGE=LYRAEXE,PIDADR=LYRAPID,-
          MBXUNT=BBUF+DI别UNIT,=.
          OUTPUT=TTNAME,QUOTA=QLIST
     BSBW   ERROR

; If okay, format an output message showing the process id...

; $FAO_S CTRSTR=PRCSTR,OUTLEN=FAOLEN,OUTBUF=FAODESC,-
     P1=LYRAPID
     BSBW   ERROR
     $OUTPUT CHAN=TTCHAN,BUFFER=FAOBUF,LENGTH=FAOLEN
     BSBW   ERROR

; Queue an I/O request to the mailbox with an AST
; to receive notification when LYRA completes.

; $QIO_S EFN=4,CHAN=EXCHAN,FUNC=$IO$_READVLK,-
     ASTADR=EXITAST,IOSB=MBXIOSB,-
     F1=EXITMSG,F2=#120
     BSBW   ERROR

..PAGE
; Place names in the group logical name table using the macro GRPNAMES.
; It will be LYRA's task, when awakened, to translate these
; names and display the results at the terminal.
; Note that translation of the name CYGNUS will require
; recursive translation.
PROGRAM EXAMPLES

PUT_NAMES:
GRNAME ORION,HUNTER
GRNAME PEGASUS,HORSE
GRNAME LYRA,HARP
GRNAME CYG,SWAN
GRNAME SWAN,DUCK
GRNAME DUCK,TALE

; After placing names in the table, wake LYRA, who has been hibernating,
; to perform the logical name translation.

$WAKE_S PIDADR=LYRAPI
BSW ERROR

; Call program DRACO (test of FAO examples in Chapter 4)
C AlleS #0,DRACO
RET ; All finished

.PAGE
; AST service routine to read the termination mailbox.
; In this example, only one message is actually expected in the mailbox
; but the program performs all the following checks:
; 1. That the I/O completed successfully.
; 2. That the message in the mailbox is a process termination message.
; 3. That the process being deleted is the subprocess created.

; This service routine enables system service failure exception
; mode as an error handling device; if a system service
; call fails, an exception condition will occur. CYGNOUS
; does not declare a condition handler, so the image
; will be forced to terminate, and the system will display
; pertinent information about the exception condition.

EXITAST:
.WORD 0 ; Entry mask
$SETSFM_S ENBFLG=1 ; Enable SSFAIL exceptions

; Check IOSB to ensure that I/O completed successfully

CMPW MBXIOSB,##SS_.NORMAL ; Check that I/O was successful
BEQ 20% ; Okay, go on
$FAO_S CSTR=ASTERNSTR=  ; Otherwise, format error msg
OUTLEN=FASTLEN,OUTBUF=FASTDESC- ; P1=IO(ERR, -
P2=MBXIOSB ; Display IOSB
$OUTPUT CHAN=TTCHAN,BUFFER=FASTBUF,LENGTH=FASTLEN
BRW 50% ; Return

; Check message type field in mailbox message to ensure that the message
; is a process termination message.
PROGRAM EXAMPLES

20$: CMNW EXITMSG+ACC+W_MSGTYPE,W_MSG+DELPROC ;Check message identification
    BEQ 30$ ;Okay, so on
    $FAQ_S CTRSTR=ASTERRSTR, - ;Otherwise, format error message
    OUTLEN=FASTLEN, OUTBUF=FASTDESC,-
    P1=IDERR, - ;Invalid PID error
    P2=EXITMSG+ACC+W_MSGTYPE ;Print message type code
    $OUTPUT CHAN=TTCHAN, BUFFER=FASTBUF, LENGTH=FASTLEN
    BRW 50$ ;Return

; Compare the second longword in the IOSB with the PID returned
; by $CREPRC to ensure that the termination message is for LYRA.

30$: CMPL LYRAID+MBPID ;LYRA deletion?
    BNEQ 35$ ;Yes, so on
    BRW 40$ ;

35$: $FAQ_S CTRSTR=PIDERRSTR, - ;Otherwise, format error message
    OUTLEN=FASTLEN, OUTBUF=FASTDESC,-
    P1=MBPID ;Display spurious PID
    $OUTPUT CHAN=TTCHAN, BUFFER=FASTBUF, LENGTH=FASTLEN
    BRW 50$ ;Return

; Format an output message indicating LYRA's final exit status
; and the time of day at which LYRA terminated.

40$: $FAQ_S CTRSTR=DONESTR, - ;Format message telling of LYRA's demise
    OUTLEN=FASTLEN, OUTBUF=FASTDESC,-
    P1=EXITMSG+ACC+W_FINALSTS, - ;Get status code
    P2=EXITMSG+ACC+W_TERMTIME ;Find time of deletion
    $OUTPUT CHAN=TTCHAN, BUFFER=FASTBUF, LENGTH=FASTLEN
    SETSFH_S ENDFLG=0 ;Disable exceptions
    RET ;Return

; This is the exit handler for CYGNUS. It receives control
; when CYGNUS exits, either normally, or as a result of
; an error condition.

EXITRTN:
    .WORD 0 ;Entry mask
    $OUTPUT CHAN=TTCHAN, BUFFER=BYTE, LENGTH=BYTELEN
    BSFW ERROR
    BLBS STATUS+20$ ;Normal exit, continue

; If error, format error message using argument list in
; exit control block

10$: $FAQ_S CTRSTR=BUFADSTR, OUTLEN=FAOLEN, OUTBUF=FAODESC,-
    P1=STATUS, P2=ERRPC
    BSFW ERROR
    $OUTPUT CHAN=TTCHAN, BUFFER=FAOBUF, LENGTH=FAOLEN

; Common code for both normal and error exit: wait for subprocess
; to terminate (if it hasn't already), then delete all names
; from the group logical name table.

B-15
20$:  \$\text{WAITFR.S EFN=\$4}
BSBW    ERROR
30$:  \$\text{DELLOG.S TBLFLG=\$1}
BSBW    ERROR
      \$\text{DASSGN.S CHAN=EXCHAN}
BSBW    ERROR
      MOVL    STATUS,RO
RET

\text{\.PAGE}

; Common error handling routine. This routine checks the
; status code in RO; if success, returns to mainline of
; program. If there is an error, the PC is placed in the exit
; control block so that exit routine can format and display
; an error message.

ERROR:
    BLBC    RO,10$
    RS8
10$: MOVL    (SP),ERRPC
    RET
\text{\.END CYGNUS}
**PROGRAM EXAMPLES**

```
.TITLE LYRA System Services Test
.IDENT /01/

; LYRA shows examples of the following system services:
; $TRNLOG - Translate Logical Name
; $ASSIGN - Assign I/O Channel
; $HIBER - Hibernate
; $FAOL - Formatted ASCII Output with List Parameter
; LYRA is the subprocess created by CYGNUM. After assigning a
; channel to its current output device, LYRA hibernates. When awakened
; by CYGNUM, LYRA translates the logical names placed in the group
; logical name table by CYGNUM, and displays the results of the
; translations on the terminal.
; When LYRA exits, a termination message is sent to the
; mailbox specified by CYGNUM.

; Macro library calls

.$$SSDEF

; Define system status values

.$$EDEF

; Local macros

; DESCRIPTOR, constructs input character string descriptors

.MACRO DESCRIPTOR TEXT, LABEL1, LABEL2
.LONG LABEL2-LABEL1
.LONG LABEL1 LABEL1: ASCII TEXT
.LABEL2: ASCII
.ENDM DESCRIPTOR

; MESSAGE, to output messages formatted by FIO

.MACRO MESSAGE
$OUTPUT CHAN=TTCCHAN, BUFFER=FALSE, LENGTH=FAOLEN
BSBW ERROR
.ENDM MESSAGE

.PAGE
.SBTL Symbols and data areas

; Local data

.PSECT RODATA, NOWRT, NOEXE
.LIST MEB

; Logical name of logical output device

OUTPUT: DESCRIPTOR <SYS$OUTPUT>

; Announcement messages

HELLO: ASCII /LYRA: INITIALIZING...AND SO TO SLEEP/
HELLOLEN: .LONG HELLOLEN-HELLO

B-17
PROGRAM EXAMPLES

WAKEMSG: .ASCII /LYRA: OKAY, WILL DO LOGICAL NAME TRANSLATION.../
WAKELEN: .LONG WAKELEN-WAKEMSG

; FAO control string for logical name output message

LOGNAMSTR:
  DESCRIPTOR <!/LYRA: AS IS A AS>

; Error message control string

ERRSTR:
  DESCRIPTOR <!/LYRA: SYSTEM SERVICE ERROR AT APP. !XL RO=!XL>

; Logical names to be translated

ORIONLOG:
  DESCRIPTOR <ORION>

CYGNUSLOG:
  DESCRIPTOR <CYGNUS>

LYRALOG:
  DESCRIPTOR <LYRA>

PEGASUSLOG:
  DESCRIPTOR <PEGASUS>

..PAGE
; Read/write data

..PSECT RWDATA, RD, WRT, NOEXE

; Output buffer for all output formatted by FAO

FAOLEN: .WORD 0
  .WORD 0
FAODESC:
  .LONG 80
  .LONG FAOBUF
FAOBUF: .BLKB 80

; Word to receive channel number of terminal

OUTCHAN: .BLKW 1

; Buffers to maintain logical name/ equivalence name pairs
; in routine that performs logical name translation

LOGBUFA:
  .LONG 63
  .LONG BUFA
BUFA: .BLKB 63
LOGBUFB:
  .LONG 63
  .LONG BUFB
PROGRAM EXAMPLES

BUFBL: .BLKB 63

LOGLEN: .LONG 0 ;Save length of equivalence name

; Parameter list for call to FAOL (used by translate routine)

TLIST:
TLOGNAM: .LONG 0 ;Address of logical name descriptor
TEQLNAM: .LONG 0 ;Address of equivalence descriptor
SAVER3: .LONG 0 ;Save register contents for switch

; Longword to store the PC when a system service call results in an
; error. LYRA checks the low bit of R0 following each service call.
; If set, LYRA continues; otherwise, it saves the PC and branches
; to an error handling routine that displays the saved PC and the
; contents of R0.

ERRPC: .LONG 0 ;For address of SSFAIL

.PAGE
.SBTLT Ready and hibernate
.PSECT CODE,EXE,RD,NOWRT
.ENABL LSB

LYRA:.
.WORD "H<2,R3,R4,R5,R6>" ;Entry mask

; Assign channel to device referred to by logical name
; SYS#OUTPUT. This name was placed in the logical name
; table by CYGNUS (it is also CYGNUS's logical output device).

20$: $ASSIGN_S DEVMAN=OUTPUT,CHAN=OUTCHAN
   BLBS R0,30$ ;Exit with status if ASSIGN fails
   RET

30$: $OUTPUT CHAN=OUTCHAN,BUFFER=HELLO,LENGTH=HELLOLEN
   BLBS R0,40$ ;Exit with status if ASSIGN fails
   MOVAL 30$+ERRPC
   BRW ERROR

40$: $HIBER_S
   BLBS R0,50$ ;Exit with status if ASSIGN fails
   MOVAL 40$+ERRPC
   BRW ERROR

50$: $OUTPUT CHAN=OUTCHAN,BUFFER=WAKEMSG,LENGTH=WAKELEN
   BLBS R0,60$ ;Exit with status if ASSIGN fails
   MOVAL 50$+ERRPC
   BRW ERROR

60$:

; When awakened, begin translating logical names. To translate the
; names, place address of a logical name descriptor in R2 and then
; go to the subroutine that performs the translation. Repeat for
; each logical name to translate.

MOVAL ORIONLOG,R2
JSB TRANSLATE
MOVAL CYGNUSLOG,R2
JSB TRANSLATE
MOVAL LYRALOG,R2
JSB TRANSLATE
MOVAL PEGASUSLOG,R2
JSB TRANSLATE

B-19
; All finished, return

RET

.PAGE
.SNTRL Subroutine to translate and print logical names
.ENABLE LS8

; On entry to this subroutine,
; R2 = address of logical name to translate
; It uses: R3 to hold address of final result buffer
; R4 to hold address of intermediate buffer

TRANSLATE:
    MOVAL LOGBUFA,R3
    MOVAL LOGBUFB,R4
    ; Get addresses of buffers

; Initial translation places resultant equivalence name in buffer pointed
; to by R3

10$:  $TRNLOG_S LOGNAM=(R2),RSLLEN=LOGLEN,RSLBUF=(R3)
    BLBS R0+30$
    MOVAL 10%+ERRPC
    BRW ERROR

; Place length of equivalence name in first word of descriptor and use this
; descriptor as input for next translation. If SS$\_NOTRAN is returned,
; then there was no recursion of name. If not, update registers to
; provide input and output descriptors for translation and repeat
; translation until SS$\_NOTRAN is returned.

30$:  MOVZWL LOGLEN,(R3)
    $TRNLOG_S LOGNAM=(R3),RSLLEN=LOGLEN,RSLBUF=(R4)
    BLBS R0+40$
    MOVAL 30%+ERRPC
    BRW ERROR

50$:  CMFW R0,#SS$\_NOTRAN
    BEQL 50$
    MOVUL R3,SAVER3
    ; Otherwise, switch
    MOVUL R4,R3
    MOVUL SAVER3,R4
    MOVZWL $63,(R4)
    ; Restore length
    BRW 30$

; Place addresses of logical name and equivalence names in FAO parameter list
; and call FAO to format output message, then output the message.

50$:  MOVUL R2,TLOGNAM
    MOVUL R3,TEQLNAM
    $FAO\_S CTRSTR=LOGNAMSTR,OUTLEN=FAOLEN,OUTBUF=FAODESC,-
    PRMLST=TLIST
    BLBS R0,60$
    MOVAL 50%+ERRPC
    BRW ERROR

60$:  $OUTPUT CHAN=OUTCHAN,BUFFER=FAOBUF,LENGTH=FAOLEN
    BLBS R0,70$
    MOVAL 60%+ERRPC
    BRW ERROR

70$:  MOVUL $63,LOGBUFA
    MOVUL $63,LOGBUFB
    ; To main routine
.PAGE
.SBTTL Error handling routine

; This routine uses the saved PC and R0 to format a message describing
; the conditions under which a call to a system service failed.

ERROR:
$FAO_S CTRSTR=ERRSTR,OUTBUF=FAODESC,OUTLEN=FAOLEN,-
       P1=ERRPC,P2=R0
$OUTPUT CHAN=OUTCHAN,BUFFER=FAOBUF,LENGTH=FAOLEN
RET
.END    LYRA
APPENDIX C
QUICK REFERENCE SUMMARY OF SYSTEM SERVICES

C.1 MACRO FORMS

C.1.1 $name_G Form
Format:

$name_G label

label
address of argument list; argument list may be created with $name macro form.

$name Macro Format:

label: $name argl,...,argn

label
symbolic address of the generated argument list.

name
macro name.

argl-argn
arguments to be placed in successive longwords in the argument list. A longword of zeros is generated for a nonspecified argument. Arguments can be specified (1) in positional order, with commas indicating no specified arguments; or (2) using keyword = argument. If keywords are used, arguments can be specified in any order.

Argument List Offset Names:

The $name macro automatically defines symbolic names for argument list of offsets. The offset names can also be defined with the $name DEF. The symbolic names defined are:

$name$ NARGS
number of arguments in list.

$name$ keyword
symbolic name for offset of each argument in list.
C.1.2 $name_S$ Form

Format:

$name_S$ arg1,...,argn

arg1 - argn
arguments for macro instruction.

Arguments can be specified (1) in positional order, with commas indicating nonspecified arguments, or (2) using keyword=argument. If keywords are used, arguments can be specified in any order.

C.2 FORTRAN FORMS

C.2.1 Procedure Call

Format:

call SYS$name$(arg1,...,argn)

arg1 - argn
arguments for system service macro instruction

Arguments must be coded in strict positional order, without keywords. Commas must be used to indicate the absence of an argument, including trailing arguments.

C.2.2 Function Reference

Format:

code = SYS$name$(arg1,...,argn)

Arguments must be coded as described above. The code and the system service function must be defined as INTEGER*4 variables.
C.3 SYSTEM SERVICE MACROS

Adjust Outer Mode Stack Pointer

$ADJSTK [acmode] , [adjust] , newadr

acmode = access mode to adjust stack pointer for
adjust = 16-bit signed adjustment value
newadr = address of longword to store updated value

Adjust Working Set Limit

$ADJWSL [pagcnt] , [wsetlm]

pagcnt = number of pages to add to working set (if positive).
Number of pages to subtract from working set (if negative).
wsetlm = address of longword to receive new working set limit,
or current working set limit if pagcnt not specified.

Allocate Device

$ALLOC devnam , [phylen] , [phybuf] , [acmode]

devnam = address of device name or logical name string descriptor
phylen = address of word to receive length of physical name
phybuf = address of physical name buffer descriptor
acmode = access mode associated with allocated device

Associate Common Event Flag Cluster

$ASCEFC efn , name , [prot] , [perm]

efn = number of any event flag in the cluster with which to associate
name = address of the text name string descriptor
prot = protection indicator for the cluster
0 -> default, any process in group
1 -> only owner's UIC
perm = permanent indicator
0 -> temporary cluster
1 -> permanent cluster
QUICK REFERENCE SUMMARY OF SYSTEM SERVICES

Convert Binary Timer to ASCII String

$ASCTIM [timlen] ,timbuf ,[timadr] ,[cvtflg]

timlen = address of a word to receive the number of characters
inserted into the output buffer.

timbuf = address of a quadword descriptor describing the
buffer to receive the converted time.

timadr = address of the quadword containing the 64-bit time to
be converted to ASCII. If 0, use current time.

cvtflg = conversion indicator
0 -> return full date and time
1 -> return converted time only

Assign I/O Channel

$ASSIGN devnam ,chan ,[acmode] ,[mbxnam]

devnam = address of device name or logical name string
descriptor

chan = address of word to receive channel number assigned

acmode = access mode associated with channel

mbxnam = address of mailbox logical name string descriptor, if
mailbox associated with device

Convert ASCII String to Binary Time

$BINTIM timbuf ,timadr

timbuf = address of string descriptor for ASCII time string

timadr = address of quadword to receive 64-bit binary time
value

Absolute time strings are specified in the format:

dd-mmm-yyyy hh:mm:ss.cc

Delta time strings are specified in the format:

dddd hh:mm:ss.cc

Broadcast

$BRDCST msgbuf ,[devnam]

msgbuf = address of message buffer string descriptor

devnam = terminal device name string descriptor. If 0, send
message to all terminals. If first word in
descriptor is 0, send message to all allocated
terminals.
Cancel I/O on Channel

$CANCEL chan
chan = number of the channel on which I/O is to be canceled

Cancel Exit Handler

$CANEXH [desblk]
desblk = address of exit control block describing exit handler to be deleted. If 0, delete all.

Cancel Timer Request

$CANTIM [reqidt],[acmode]
reqidt = request identification for request to be canceled. If 0, all requests canceled.
acmode = access mode of requests to be canceled

Cancel Wakeup

$CANWAK [pidadr],[prcnam]
pidadr = address of process identification of process for which wakeups are to be canceled
prcnam = address of process name string descriptor

Clear Event Flag

$CLREF efn
efn = number of event flag to be cleared

Change to Executive Mode

$CMEXEC routin ,[arglst]
routin = address of the routine to be executed in executive mode
arglst = address of argument list to be supplied to the routine

Change to Kernel Mode

$CMKRNL routin ,[arglst]
routin = address of routine to be executed in kernel mode
arglst = address of argument list to be supplied to routine
QUICK REFERENCE SUMMARY OF SYSTEM SERVICES

Contract Program/Control Region

\$CNTREG pagcnt,[retadr],[acmode],[region]

pagcnt = number of pages to be deleted from end of region

retadr = address of 2-longword array to receive virtual addresses of starting and ending page of deleted area

acmode = access mode for which service is performed

region = region indicator
  0 -> program (P0) region
  1 -> control (P1) region

Create Logical Name

\$CRELOG [tblflg],lognam,eqlnam,[acmode]

tblflg = logical name table number
  0 -> system (default)
  1 -> group table
  2 -> process table

lognam = address of logical name string descriptor

eqlnam = address of equivalence name string descriptor

acmode = access mode for logical name (process table only)

Create Mailbox and Assign Channel

\$CREMBX [prmflg],chan,[maxmsg],[bufquo],[promsk],[acmode],
[lognam]

prmflg = permanent flag
  0 -> temporary mailbox (default)
  1 -> permanent mailbox

chan = address of word to receive channel assigned

maxmsg = maximum message size that may be received by mailbox

bufquo = number of bytes of dynamic memory that can be used to buffer mailbox messages

promsk = protection mask for mailbox

acmode = access mode of created mailbox

lognam = address of logical name string descriptor for mailbox
QUICK REFERENCE SUMMARY OF SYSTEM SERVICES

Create Process

$CREPRC [pidadr] ,[image] ,[input] ,[output]
, [error] ,[prvadr] ,[quota] ,[prcnam]
,[baspri] ,[uic] ,[mbxunt] ,[stsflg]

pidadr = address of longword in which to return process identification of created process

image  = address of string descriptor for image name

input  = address of string descriptor for SYS$INPUT logical name

output = address of string descriptor for SYS$OUTPUT logical name

error  = address of string descriptor for SYS$ERROR logical name

prvadr = address of quadword privilege list

quota  = address of quota list

prcnam = address of string descriptor for process name

baspri = base priority (0-31) to set for new process (macro default = 2)

uic    = user identification code. If 0, create a subprocess

mbxunt = mailbox unit for termination message

stsflg = status and mode flag bits

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>disable resource wait mode</td>
</tr>
<tr>
<td>1</td>
<td>enable system service failure exception mode</td>
</tr>
<tr>
<td>2</td>
<td>inhibit process swapping</td>
</tr>
<tr>
<td>3</td>
<td>disable accounting messages</td>
</tr>
<tr>
<td>4</td>
<td>batch process</td>
</tr>
<tr>
<td>5</td>
<td>cause created process to hibernate</td>
</tr>
<tr>
<td>6</td>
<td>allow login without authorization file check</td>
</tr>
<tr>
<td>7</td>
<td>process is a network connect object</td>
</tr>
</tbody>
</table>

Create Virtual Address Space

$CRETVA inadr ,[retadr] ,[acmode]

inadr  = address of 2-longword array containing starting and ending virtual address of pages to be created

retadr = address of a 2-longword array to receive starting and ending virtual address of pages actually created

acmode = access mode for the new pages (protection is read/write for acmode and more privileged modes)
QUICK REFERENCE SUMMARY OF SYSTEM SERVICES

Create and Map Section

$CRMPSC [inadr],[retadr],[acmode],[flags],[gsdnam],[ident],[replpag],[chan],[pagcnt],[vbn],[prot],[pfc]

inadr = address of 2-longword array containing starting and
ending virtual addresses of space into which section
is to be mapped

retadr = address of 2-longword array to receive addresses
actually mapped

acmode = access mode of owner of pages

flags = section characteristics

<table>
<thead>
<tr>
<th>Flag</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEC$M_GBL</td>
<td>Global section</td>
</tr>
<tr>
<td>SEC$M_CRF</td>
<td>Copy-on-reference pages</td>
</tr>
<tr>
<td>SEC$M_DZRO</td>
<td>Demand zero pages</td>
</tr>
<tr>
<td>SEC$M_WRT</td>
<td>Read/write section</td>
</tr>
<tr>
<td>SEC$M_PERM</td>
<td>Permanent section</td>
</tr>
<tr>
<td>SEC$M_SYSGBL</td>
<td>System global section</td>
</tr>
</tbody>
</table>

gsdnam = address of global section name string descriptor

ident = address of quadword containing version identification
and match control

replpag = relative page number within section to begin mapping

chan = number of channel on which file is accessed

pagcnt = number of pages in section

vbn = virtual block number of beginning of section

prot = protection mask

pfc = page fault cluster size

Disassociate Common Event Flag Cluster

$DACEFC efn

efn = number of any event flag in the cluster to be
disassociated

Deallocate Device

$DALLOC [devnam],[acmode]

devnam = address of device name string descriptor. If 0,
deallocate all devices.

acmode = access mode associated with device
Deassign I/O Channel

$DASSGN chan

chan = number of channel to be deassigned

Declare AST

$DCLAST astadr ,[astprm] ,[acmode]

astadr = address of entry mask of AST routine
astprm = value to be passed to AST routine as an argument
acmode = access mode for which the AST is to be declared

Declare Change Mode or Compatibility Mode Handler

$DCLCMH addr ,[prvhd] ,[type]

addr = address of change mode or compatibility mode handler
prvhd = address of longword to receive previous handler address

type = handler type indicator
0 -> change mode handler for current mode
1 -> compatibility mode handler

Declare Exit Handler

$DCLEXH desblk

desblk = address of exit control block containing:

<table>
<thead>
<tr>
<th>31</th>
<th>8 7</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>forward link</td>
<td></td>
<td></td>
</tr>
<tr>
<td>exit handler address</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>address to store reason for exit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>additional arguments for exit handler, if any</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Delete Logical Name

$DELLOG [tblflg], [lognam], [acmode]

tblflg = logical name table number
0 -> system
1 -> group
2 -> process

lognam = address of logical name string descriptor. If 0, delete all names in the specified table.

acmode = access mode of logical name (process table only)

Delete Mailbox

$DELMBX chan

chan = channel number assigned to the mailbox

Delete Process

$DELPRC [pidadr], [prcnam]

pidadr = address of longword containing process identification of process to be deleted

prcnam = address of string descriptor for process name of process to be deleted.

Delete Virtual Address Space

$DELTVA inadr, [retadr], [acmode]

inadr = address of 2-longword array containing starting and ending virtual addresses of pages to delete

retadr = address of 2-longword array to receive starting and ending addresses of pages actually deleted

acmode = access mode for which service is performed

Delete Global Section

$DBGLSC [flags], [gsdnam], [ident]

flags = type of section
0 -> group global section
SEC$M_SYSGBL -> system global section

gsdnam = address of global section name string descriptor

ident = address of quadword containing version identification and match control

Delete Common Event Flag Cluster

$DLCEFC name

name = address of text name string descriptor of permanent cluster
Exit

$EXIT [code]

code = longword to be saved in process header as completion status of current image (macro default = 1)

Expand Program/Control Region

$EXPREG pagcnt,[retadr],[acmode],[region]

pagcnt = number of pages to add to end of specified region

retadr = address of 2-longword array to receive virtual addresses of starting and ending pages of expanded region

acmode = access mode of the new pages

region = region indicator
0 -> expand program (P0) region
1 -> expand program (P1) region

Formatted ASCII Output

$FAO ctrstr,[outlen],outbuf,[p1],[p2]...[pn]

ctrstr = address of string descriptor for ASCII control string

outlen = address of word in which to store output string length

outbuf = address of output buffer string descriptor

pl... = variable number of arguments to FAO

Formatted ASCII Output With List Parameter

$FAOL ctrstr,[outlen],outbuf,prmlst

ctrstr = address of string descriptor for control string

outlen = address of word to receive output string length

outbuf = address of output buffer string descriptor

prmlst = address of a list of longword parameters

Force Exit

$FORCEX [pidadr],[prcnam],[code]

pidadr = address of process identification of process to be forced to exit

prcnam = address of process name string descriptor for forced process

code = longword completion status for exit service
QUICK REFERENCE SUMMARY OF SYSTEM SERVICES

Get I/O Channel Information

SGETCHN chan,[prilen],[probuf],[scdlen],[scdbuf]
chan = number of a channel assigned to the device
prilen = address of word to receive length of primary buffer
probuf = address of primary buffer descriptor
scdlen = address of word to receive length of secondary buffer
scdbuf = address of secondary buffer descriptor

Get I/O Device Information

SGETDEV devnam,[prilen],[probuf],[scdlen],[scdbuf]
devnam = address of device name or logical name string descriptor
prilen = address of word to receive length of primary buffer
probuf = address of primary buffer descriptor
scdlen = address of word to receive length of secondary buffer
scdbuf = address of secondary buffer descriptor

Get Job/Process Information

SGETJPI,[pidadr],[prcnam],[itmlst],
pidadr = address of process identification
prcnam = address of process name string descriptor
itmlst = address of a list of item descriptors
First, and fifth through seventh arguments are reserved

Get Message

SGETMSG msgid,msglen,bufadr,[flags],[outadr]
msgid = identification of message to be retrieved
msglen = address of a word to receive length of string returned
bufadr = address of buffer descriptor of buffer to receive string
flags = flag bits for message content (macro default = 15)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>Include text</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>Do not include text</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Include identifier</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>Do not include identifier</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>Include severity</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>Do not include severity</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>Include component</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>Do not include component</td>
</tr>
</tbody>
</table>

outadr = address of 4-byte array to receive

<table>
<thead>
<tr>
<th>Byte</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>Count of FAO arguments</td>
</tr>
<tr>
<td>2</td>
<td>User value</td>
</tr>
<tr>
<td>3</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

Get Time

$GETTIM timadr

timadr = address of a quadword to receive 64-bit current time value

Hibernate

$HIBER_S

$INPUT Macro

$INPUT chan ,length ,buffer ,[iosb] ,[efn]

chan = number of the channel on which I/O is to be performed
length = length of the input buffer
buffer = address of the input buffer
iosb = address of quadword I/O status block
efn = event flag to wait on (default = 0)

Lock Pages in Memory

$LCKPAG inadr ,[retadr] ,[acmode]

inadr = address of 2-longword array containing starting and ending addresses of pages to be locked
retadr = address of 2-longword array to receive addresses of pages actually locked
acmode = access mode to check against the owner of the pages
Lock Pages in Working Set

\$LKASET inadr , [retadr] , [acmode]

\textit{inadr} = address of 2-longword array containing starting and ending virtual addresses of pages to be locked

\textit{retadr} = address of a 2-longword array to receive starting and ending virtual addresses of pages actually locked

\textit{acmode} = access mode to be checked against the page owner

Map Global Section


\textit{inadr} = address of 2-longword array containing starting and ending addresses of pages to be mapped

\textit{retadr} = address of 2-longword array to receive virtual addresses of pages mapped

\textit{acmode} = access mode of owner of mapped pages

\textit{flags} = flags overriding default section characteristics

<table>
<thead>
<tr>
<th>Flag</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{SEC$M_WRT}</td>
<td>Read/write section</td>
</tr>
<tr>
<td>\textit{SEC$M_SYSGL}</td>
<td>System global section</td>
</tr>
</tbody>
</table>

\textit{gsdnam} = address of global section name descriptor

\textit{ident} = address of quadword containing version identification and match control

\textit{replag} = relative page number within global section

Convert Time to Numeric

\$NUMTIM timbuf , [timadr]

\textit{timbuf} = address of a 7-word buffer to receive numeric time information

\textit{timadr} = address of a quadword containing the 64-bit time. If 0, use current time

Buffer format:

<table>
<thead>
<tr>
<th>31</th>
<th>16 15</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>month of year</td>
<td>year since 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>hour of day</td>
<td>day of month</td>
<td></td>
<td></td>
</tr>
<tr>
<td>second of minute</td>
<td>minute of hour</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>hundredths of second</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

C-14
$OUTPUT Macro

$OUTPUT chan, length, buffer, [iosb], [efn]
chan = channel on which I/O is directed
length = length of the output buffer
buffer = address of the output buffer
iosb = address of quadword I/O status block
efn = event flag number to wait (default = 0)

Purge Working Set

$PURGWS inadr

inadr = address of 2-longword array containing starting and ending addresses of pages to be removed

Put Message

$PUTMSG msgvec ,[actrtn] ,[facnam]
msgvec = address of message argument vector
actrtn = address of entry mask of action routine
facnam = address of facility name string descriptor

Queue I/O Request

$QIO [efn] ,chan ,func ,[iosb] ,[astadr] ,[astprm]

efn = number of event flag to set on completion
chan = number of channel on which I/O is directed
func = function code specifying action to be performed
iosb = address of quadword I/O status block to receive final completion status information
astadr = address of entry mask of AST routine
astprm = value to be passed to AST routine as argument
pl... = optional device- and function-specific parameters

Queue I/O Request and Wait for Event Flag

See QIO for argument description

C-15
Quick Reference Summary of System Services

Read Event Flag

$REDEFINE  efn , state

efn  = event flag number of any flag in the cluster to be read
state = address of a longword to receive current state of all flags in the cluster

Resume Process

$RESUME  [pidadr] , [prcnam]

pidadr = address of process identification of process to be resumed
prcnam = address of process name string descriptor

Schedule Wakeup

$SCHEDWK  [pidadr] , [prcnam] , daytim , [reptim]

pidadr = address of process identification of process to be awakened
prcnam = address of process name string descriptor
daytim = address of quadword containing time to wake
reptim = address of quadword containing repeat time interval

Set AST Enable

$SETAST  enbflg

enbflg = AST enable indicator
  0 -> disable ASTs for caller at current access mode
  1 -> enable ASTs for caller at current access mode

Set Event Flag

$SETEF  efn

efn  = event flag number of flag to set

Set Exception Vector

$SETEXV  [vector] , [address] , [acmode] , [prvhnld]

vector = vector number
  0 -> modify primary vector
  1 -> modify secondary vector
  2 -> modify last chance vector

address = exception handler address (0 indicates deassign vector)

acmode = access mode for which vector is set

prvhnld = address of longword to receive previous handler address
**QUICK REFERENCE SUMMARY OF SYSTEM SERVICES**

Set Timer

```c
$SETIMR [efn],daytim,[astadr],[reqidt]
```

- **efn** = event flag to set when timer expires
- **daytim** = address of quadword containing 64-bit time value
- **astadr** = address of entry mask of AST routine
- **reqidt** = request identification of this timer request

Set Power Recovery AST

```c
$SETPRA astadr,[acmode]
```

- **astadr** = address of power recovery AST routine
- **acmode** = access mode of AST

Set Priority

```c
$SETPRI [pidadr],[prcnam],pri,[prvpri]
```

- **pidadr** = address of process identification of process to set priority for
- **prcnam** = address of process name string descriptor
- **pri** = new base priority for the process 0 - 15 are background, 16 - 31 are time-critical
- **prvpri** = address of longword to receive previous base priority

Set Process Name

```c
$SETPRN [prcnam]
```

- **prcnam** = address of the process name string descriptor

Set Protection on Pages

```c
$SETPRT inadr,[retadr],[acmode],prot,[prvprrt]
```

- **inadr** = address of 2-longword array containing starting and ending virtual addresses of pages to change protection for
- **retadr** = address of 2-longword array containing starting and ending addresses of pages which had their protection changed is returned
- **acmode** = access mode of request
- **prot** = new protection
- **prvprrt** = address of byte to receive previous protection of last page changed
QUICK REFERENCE SUMMARY OF SYSTEM SERVICES

Set Resource Wait Mode

$SETRWM [watflg]

watflg = wait indicator
0 -> wait for resources
1 -> return failure status immediately

Set System Service Failure Mode

$SETSPM [enbflg]

enbflg = enable indicator
0 -> disable generation of exceptions on system service failures
1 -> generate exceptions for system service failures

Set Process Swap Mode

$SETSWM [swpflg]

swpflg = swap indicator
0 -> enable swapping
1 -> disable swapping (lock in balance set)

Send Message to Accounting Manager

$SNDACC msgbuf,[chan]

msgbuf = address of message buffer string descriptor
chan = number of channel assigned to mailbox to receive reply

Send Message to Error Logger

$SNDERR msgbuf

msgbuf = address of message buffer string descriptor

Send Message to Operator

$SNDOPR msgbuf,[chan]

msgbuf = address of message buffer string descriptor
chan = number of channel assigned to mailbox to receive reply

Send Message to Symbiont Manager

$SNDSMB msgbuf,[chan]

msgbuf = address of message buffer string descriptor
chan = number of channel assigned to mailbox to receive reply
Suspend Process

\$SUSPND [pidadr] ,[prcnam]

pidadr = address of process identification of process to suspend
prcnam = address of process name string descriptor

Translate Logical Name

\$TRNLOG lognam ,[rsllen] ,rslbuf ,[table] ,[acmode] ,[dsbmsk]

lognam = address of logical name string descriptor
rsllen = address of word to receive length of resultant name string
rslbuf = address of result string buffer descriptor
table = address of byte to receive logical name table number
acmode = address of byte to receive access mode of entry (process table only)
dsbmsk = table search disable mask

<table>
<thead>
<tr>
<th>Bit Set</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Do not search system</td>
</tr>
<tr>
<td>1</td>
<td>Do not search group</td>
</tr>
<tr>
<td>2</td>
<td>Do not search process</td>
</tr>
</tbody>
</table>

Unlock Pages From Memory

\$ULKPAG inadr ,[retadr] ,[acmode]
inadr = address of 2-longword array containing starting and ending virtual addresses of pages to be unlocked
retadr = address of a 2-longword array to receive starting and ending virtual addresses of pages actually unlocked
acmode = access mode to check against the owner of the pages

Unlock Pages From Working Set

\$ULWSET inadr ,[retadr] ,[acmode]
inadr = address of 2-longword array containing starting and ending virtual addresses of pages to be unlocked
retadr = address of a 2-longword array to receive starting and ending virtual addresses of pages actually unlocked
acmode = access mode to check against the owner of the pages
QUICK REFERENCE SUMMARY OF SYSTEM SERVICES

Unwind Call Stack

$UNWIND [depadr], [newpc]

depadr = address of number of logical frames to unwind call stack

newpc = change of flow PC

Update Section File on Disk

$UPDSEC inadr, [retadr],[acmode],[updflg],[efn],[iosb], [astadr],[astprm]

inadr = address of 2-longword array containing starting and ending addresses of the pages to be potentially written

retadr = address of 2-longword array to receive addresses of the first and last page queued in the first I/O request

acmode = access mode on behalf of which the service is performed

updflg = update indicator for writable global sections
0 -> write all read/write pages in the section
1 -> write all pages modified by the caller

efn = number of event flag to set when the section file is updated

iosb = address of quadword I/O status block

astadr = address of entry mask of an AST service routine

astprm = AST parameter to be passed to the AST service routine

Wait for Single Event Flag

$WAITFR efn

efn = event flag number of event to wait for

Wake

$WAKE [pidadr], [prcnam]

pidadr = address of process identification of process to be awakened

prcnam = address of process name string descriptor
Wait for Logical AND of Event Flags

$WFLAND efn ,mask

efn  = event flag number of any flag within the cluster
mask = 32-bit mask of flags that must be set

Wait for Logical OR of Event Flags

$WFLOR efn ,mask

efn  = event flag number of any flag within the cluster
mask = 32-bit mask of flags, any of which must be set
INDEX

$ACDEFF macro, symbols defined, 4-39
   process termination message offsets, 4-39
$ADJSKT format, 4-3
$ADJWSL format, 4-5
$ALLOC format, 4-6
$ASCEFC format, 4-8
$ASCTIM format, 4-10
$ASSIGN format, 4-12
$BINTIM format, 4-15
$BRDCST format, 4-17
$CANCEL format, 4-19
$CANEXH format, 4-21
$CANTIM format, 4-22
$CANWAR format, 4-23
$CHDFDEF macro, 3-67
$CLREF format, 4-25
$CMEXEC format, 4-26
$CMKRNL format, 4-27
$CNTREG format, 4-28
$CRELOG format, 4-30
$CREMBX format, 4-32
$CREPRC format, 4-35
$CRETVA format, 4-44
$CRMSPSC format, 4-46
$DACEFC format, 4-52
$DALLOC format, 4-53
$DASSGN format, 4-55
$DCLAST format, 4-57
$DCLCMH format, 4-58
$DCLEXH format, 4-60
$DELLOG format, 4-62
$DELMBX format, 4-64
$DELPRC format, 4-66
$DELTVA format, 4-68
$DBGCSCL format, 4-70
$DIBDEF macro, symbols defined, 4-93
$DLCEFC format, 4-72
$EXIT format, 4-73
$EXPREG format, 4-74
$FAO format, 4-76
$FAOL format, 4-77
$FORCEX format, 4-90
$GETCHN format, 4-92
$GETDEV format, 4-95
$GETJPI format, 4-97
$GETMSG format, 4-102
$GETTIM format, 4-104
$HIBER format, 4-105
$INPUT macro, example, 3-25 format, 4-106
$IODEF macro, 3-22 symbols defined, A-6
$JBCMSGDEF macro, 4-152, 4-167
$JPIDEF macro, 4-98 symbols defined, 4-100
$LCKPAG format, 4-107
$LKWSET format, 4-109
$MGBLSC format, 4-111
$MSGDEF macro, A-7 symbolic names defined, A-7
$nameDEF macro, 2-5
$name_G form of system service macro, 2-3 example, 2-4
$name_S form of system service macro, 2-6 example, 2-8
$NUMTIM format, 4-114
$OPCDEF macro, 4-155 symbols defined, 4-155
$OUTPUT macro, example, 3-25 format, 4-116
$PQLDEF macro, 4-40 symbols defined, 4-41
$PRDEF macro, A-7 symbols defined, A-7
$PRRTDEF macro, 4-143 symbols defined, A-7
$PRVDEF macro, 4-36 symbols defined, 4-36
$PSLDEF macro, A-8 symbols defined, A-8
$PURGWS format, 4-117
$PUTMSG format, 4-118
$QIO format, 4-124
$QIOW,
   $INPUT and $OUTPUT forms, 3-25 format, 4-127
$READEF format, 4-128
$RESUME format, 4-129
$SCHDWK format, 4-131
$SECDDEF macro, 4-46 symbols defined, 4-46, 4-112
$SETAST format, 4-133
$SETDEF format, 4-134
$SETEXV format, 4-135
$SETIM format, 4-137
$SETPMRA format, 4-139
$SETPRI format, 4-140
$SETPRN format, 4-142
$SETPT format, 4-143
$SETWM format, 4-145
$SETSFM format, 4-146
$SETSWM format, 4-147
$SMRDEF macro, 4-160 symbols defined, 4-163
$SNDBACC format, 4-148
$SNDRERR format, 4-153
$SNDFPR format, 4-154
$SNDSMB format, 4-159
$SSDEF macro, 2-11 symbols defined, A-8
$SUSPND format, 4-169

Index-1
INDEX (Cont.)

$TRNLOG format, 4-171
$ULKPAK format, 4-173
$ULMSET format, 4-175
$UNWIND format, 4-177
$UDPSEC format, 4-179
$WAITFTR format, 4-182
$WAKE format, 4-183
$WFLAND format, 4-185
$WFLOR format, 4-186

A
Absolute time, 3-56
  buffer format, 4-15
Access modes, 1-2
  conventions for coding, 2-10,
  2-18
  effect on AST delivery, 3-14
  symbolic names defined, A-8
Accounting file, 4-148
  format of records, 4-151
ACP interface driver I/O
  function codes, A-6
Addresses,
  virtual, 3-79
Adjust Outer Mode Stack Pointer
  ($ADJSTK) system service, 4-3
Adjust Working Set Limit
  ($AJWLSL) system service, 4-5
  increase working set size, 3-81
Allocate Device ($ALLOC) system
  service, 4-6
  example, 3-29
Allocation,
  device, 3-28, 4-6
Argument list, 2-2
  for AST service routine, 3-12
  for system services,
  format, 2-2
  passed to a condition handler, 3-67
Arguments, 2-17
  FORTRAN coding summary, 2-17
Arrays,
  argument lists for condition
  handlers, 3-68
  virtual address, 3-80
ASSIGN command, 3-15
Assign I/O Channel ($ASSIGN)
  system service, 4-12
  example, 3-21
Associate Common Event Flag
  Cluster ($ASCEFC) system
  service, 4-8
  examples, 3-7, 3-9
AST, 3-10
  declare, 4-57
  example, 3-14
AST (Cont.),
  delivery, 3-13
  disable/enable delivery, 4-133
  execution,
    access modes, 3-12
    power recovery, 4-139
    service routine, 3-12
    example, 3-13
  services,
    general information, 3-10
    summary, 1-3
    synchronize I/O completion,
    3-23
    used with timer services,
    3-58
    example, 3-60

B
Balance set, 3-82, 4-147
  swapping, 3-82
Broadcast ($BRDCST) system
  service, 4-17

C
Cancel Exit Handler ($CANEXH)
  system service, 3-49, 4-21
Cancel I/O On Channel ($CANCEL)
  system service, 3-27, 4-19
  example, 3-27
Cancel Timer Request ($CANTIM)
  system service, 4-22
  examples, 3-60
Cancel Wakeup ($CANWAK) system
  service, 4-23
  cancel wakeup requests, 3-61
Card reader driver I/O func-
  tion codes, A-5
Change mode,
  handler, 3-64, 4-58
  services,
    summary, 1-5
    to executive, 4-26
    to kernel, 4-27
Change to Executive Mode
  ($CMEXEC) system service, 4-26
Change to Kernel Mode ($CMKRN)
  system service, 4-27
Channel assignment, 3-21, 4-12
  mailboxes, 4-32
Character string descriptor,
  2-18
  FORTRAN coding, 2-18
  MACRO coding, 2-9
  macro to generate, 2-10

Index-2
Clear Event Flag ($CLREF)  
  system service, 4-25  
  example, 3-6  
Clusters,  
  event flag, 3-4  
Common event flag cluster, 3-7,  
  4-8  
  for process communication,  
  3-44  
Compatibility mode handler,  
  3-64, 4-58  
Condition handler, 3-63  
  courses of action, 3-69  
  declare on call stack, 3-64  
  example of condition handling  
  routines, 3-72  
  search of call stack, 3-66  
Condition handling services,  
  3-63  
  general information, 3-63  
  summary, 1-5  
Contract Program/Control Region  
  ($CNTREG) system service,  
  4-28  
  example, 3-78  
Control block,  
  exit handler, 4-60  
Control region, 3-77  
  contract, 4-28  
  expand, 4-74  
Control string,  
  FAO, 4-78  
Conventions for coding, 2-11  
  access modes, 2-11  
  arguments to system services,  
  MACRO, 2-8  
  FORTRAN, 2-16  
Convert ASCII String to Binary  
  Time ($BINTIM) system  
  service, 4-15  
  examples, 3-57, 3-58  
Convert Binary Time to ASCII  
  String ($ASCTIM) system  
  service, 4-10  
  example, 3-57  
Convert Binary Time to Numeric  
  Time ($NUMTIM) system  
  service, 4-114  
Create and Map Section ($SCRMPSC)  
  system service, 4-46  
  example of mapping a section,  
  3-86  
Create Logical Name ($CRELOG)  
  system service, 4-30  
  example, 3-15  
Create Mailbox and Assign  
  Channel ($CREMBOX) system  
  service, 4-32  
  examples, 3-34, 3-54  
Create Process ($CREPRC) system  
  service, 4-35  
  examples, 3-38  
Create Virtual Address Space  
  ($CRETVA) system service,  
  4-44  

D  
Date,  
  system format, 3-56  
Deallocate Device ($DALLOC)  
  system service, 3-29, 4-53  
Deassign I/O Channel ($DASSGN)  
  system service, 4-55  
  example, 3-25  
Declare AST ($DCLAST) system  
  service, 4-57  
  example, 3-14  
Declare Change Mode or Compati- 
  bility Mode Handler ($DCLCMH)  
  system service, 4-58  
Declare Exit Handler ($DCLEXH)  
  system service, 4-60  
  example, 3-50  
Default,  
  arguments for system  
  services, 2-8, 2-14  
  disk and directory for  
  created process, 3-40  
Delete Common Event Flag  
  Cluster ($DLCFC) system  
  service, 4-72  
Delete Global Section ($DGBLSC)  
  system service, 4-70  
Delete Logical Name ($DELLLOG)  
  system service, 4-62  
Delete Mailbox ($DELMBOX) system  
  service, 4-64  
Delete Process ($DELPRC) system  
  service, 3-53, 4-66  
Delete Virtual Address Space  
  ($DELTVA) system service,  
  4-68  
  example, 3-78  
Delete,  
  common event flag clusters,  
  3-8, 4-72  
  mailboxes, 3-33, 4-64  
  processes, 3-53, 4-66  
  timer requests, 3-60, 4-22  
  virtual address space, 3-78,  
  4-68  
Delivery, 3-13  
  AST,  
  enable/disable, 4-133  
Delta time, 3-56  
  how to specify, 3-58  

Index-3
INDEX (Cont.)

DESCRIPTOR macro, 2-10
Descriptor,
character string,
FORTRAN coding, 2-19
MACRO coding, 2-9
Detached process, 3-41
compared with subprocess,
3-37
Device,
allocate, 3-28, 4-6
assign I/O channel, 3-21, 4-12
deallocate, 3-29, 4-53
information, 4-92, 4-95
names, 3-31
physical names vs. logical
names, 3-29
Directive (FAO),
format, 4-78
summary, 4-80
Disassociate Common Event Flag
Cluster ($DACEF$) system
service, 4-52
example, 3-7
Disk driver I/O function codes,
A-3
Dispatcher,
exception, 3-65
DMCII driver I/O function codes,
A-5

Event flag (Cont.),
services,
genernal information, 1-3,
3-4
summary, 1-3
setting, 4-134
used with I/O services, 3-23
used with timer services, 3-58
example, 3-59
waits, 3-5, 3-11
Exception, 4-146
caused by system service
failure, 4-146
conditions, 3-63
summary, 3-70
dispatcher, 3-64
vectors, 3-64, 4-135
Exit ($EXIT$) system service,
4-73
cause image exit, 3-48
Exit, 4-90
forced, 4-90
handler, 4-60
cancel, 4-21
control block format, 4-60
example, 3-50
image exit, 3-47
Expand Program/Control Region
($EXPRG$) system service,
4-74
example, 3-78

E

Equivalence names, 3-15
Error,
cause exception condition,
2-13
checking,
FORTRAN, 2-22
MACRO, 2-12
logger,
send message to, 4-153
messages,
obtain text, 4-102
output, 4-118
return status codes, 2-11,
2-20
stream defined for process,
3-39
Event flag, 3-4
clearing, 4-25
clusters, 3-4
common clusters, 3-7
associate, 3-7, 4-8
create, 3-7, 4-8
delete, 4-72
disassociate, 4-52
read status of, 4-128

F

FAO, 3-31, 4-76
control string, 4-78
directives,
examples, 4-83
format, 4-78
summary, 4-80
Force Exit ($FORCEX$) system
service, 4-90
contrast with process
deletion, 3-53
Formatted ASCII Output ($FAO$)
system service, 4-76
examples, 3-32, 4-84
Formatted ASCII Output with
List Parameter ($FAOL$)
macro, 4-77
example, 4-85
FORTRAN,
coding system service calls,
2-14, 2-15
function reference, 2-15
procedure call, 2-15

Index-4
INDEX (Cont.)

Function codes for I/O operations, 3-22
summary, A-3
Function reference, 2-15
FORTRAN system service call, 2-15

G

Get I/O Channel Information
(SGETCHN) system service, 4-92
example, 3-54
Get I/O Device Information
(SGETDEV) system service, 4-95
Get Job/Process Information
(SGETJPI) system service, 4-97
used for process control,
3-44, 3-86
Get Message (SGETMSG) system
service, 4-102
Get Time (SGETTIM) system
service, 3-57, 4-104
Global sections, 4-70
creating, 4-46
defined, 3-82
deleting, 4-70
group and system, 3-85
mapping, 3-87, 4-46, 4-111
temporary and permanent, 3-85
Group, 3-16
logical name table, 3-16
number,
qualify process names, 3-43
restrict system service
use, 1-2

H

Handler,
change mode, 4-58
compatibility mode, 4-58
condition, 3-63
exit, 3-49, 4-60
cancel, 4-21
Hibernate ($HIBER) system
service, 4-105
example, 3-46
Hibernate, 3-45
compared with suspension,
3-45
schedule wakeup, 3-61

I

I/O,
$QIO system service, 4-124
$QIOW system service, 4-127
cancel, 3-27, 4-19
channels,
assign, 3-21, 4-12
deassign, 4-55
obtain information, 4-92
device,
obtain information, 4-95
example, 3-26
function codes,
how used, 3-22
summary, A-3
mailboxes,
example, 3-34
services,
genral information, 3-21
summary, 4-4
status block, 3-24, 4-126
Image,
exit, 3-47, 4-73
compared with process
deletion, 3-55
exit handlers, 3-49
forced, 3-50, 3-53
rundown, 3-48
force exit, 3-50, 3-53, 4-90
Indicators,
conventions for coding, 2-10,
2-18
Input,
stream defined for process,
3-39
terminus I/O, 3-26
virtual blocks, 4-106

L

Line printer driver I/O
function codes, A-4
Lock Pages in Memory ($LKCPAG)
system service, 4-107
Lock Pages in Working Set
($LKWSWET) system service,
4-109
increase program efficiency,
3-81
Lock pages,
memory, 3-82
working set, 3-81
Logical names, 3-15
create, 4-30
example, 3-15

Index-5
INDEX (Cont.)

Logical names (Cont.),
   delete, 3-19, 4-62
   process permanent files, 3-19
   services,
      general information, 3-15
      summary, 1-4
   tables, 3-16
      example, 3-17
   translation, 3-18, 4-171
   used by I/O services, 3-29
   used for process communication, 3-44

M

Magnetic tape driver I/O
   function codes, A-4
Mailbox driver I/O function
   codes, A-5
Mailboxes, 3-33, 4-32
   creating, 4-32
   deleting, 4-64
   example of creation and I/O, 3-34
   used for process communication, 3-44
   used for process termination message, 3-54
Map Global Section (SMGBLSC)
   system service, 4-111
   example, 3-87
Mapping, 3-87
   global sections, 3-87, 4-46, 4-111
   sections, 3-83, 3-85
Maximize access mode,
   definition, 2-11
Memory,
   lock pages, 3-82, 4-107, 4-109
   management services,
      general information, 3-77
      summary, 1-5
   unlock pages, 4-173
Messages,
   associated with system status codes, 2-12, 4-102
   output, 4-118

N

NARGS, 2-5
Numeric time buffer format, 4-114

O

Open,
   disk file for use as a section, 3-83
Operator,
   send message to, 4-154
Output,
   format character strings, 4-76
   formatting with $FAO, 3-31
   stream defined for process, 3-39
   system messages, 4-118
   virtual blocks, 4-116
Owner,
   of memory page, 3-80

P

Page, 3-77, 3-85
   copy-on-reference,
      define in section, 3-85
   demand-zero,
      define in section, 3-85
   lock in memory, 4-107
   lock in working set, 4-109
   protection,
      set or change, 4-143
   symbolic names, A-7
Paging,
   sections, 3-89
   working set, 3-81
Parameter,
   FAO, 4-78
   for AST service routine, 3-13
Print queue,
   manipulate, 4-159
Priority,
   set or change process, 4-140
Private sections, 3-82
   creating and mapping, 4-46
Privilege,
   defined by access mode, 1-2
   defined for process, 3-41, 4-35
   required for process control, 3-41
   to use system services, 1-1
Process, control services,
   general information, 3-37
   summary, 1-4
   creation, 4-35
   example, 3-42
   deletion, 3-51, 4-66
   compared with image exit, 3-55

Index-6
Process (Cont.),
detached process, 3-37
identification, 3-42
logical name table, 3-16
name, 3-42
qualified by group number, 3-43
set or change, 4-142
obtain information, 4-97
permanent files, 3-19
resume after suspension, 4-129
set or change priority, 4-140
subprocess, 3-37
suspend, 4-169
termination message format, 4-39
Processor registers,
symbolic names, A-7
Processor status longword,
symbolic field definitions, A-8
Program region, 3-77, 3-78
contract, 4-28
example of expanding, 3-78
expand, 4-74
Protection,
page, 4-143
Purge Working ($SPURGWS) system service, 4-117
Put Message ($PUTMSG) system service, 4-118

Queue I/O Request ($QIO) system service, 4-124
example, 3-22
Queue I/O Request and Wait for Event Flag ($QIOW) system service, 4-127
Quotas, 4-40

R
Read Event Flags ($READEF) system service, 4-128
Resource,
quotas, 1-1, 4-40
wait mode, 2-13
set or change, 4-145
Resume Process ($RESUME) system service, 4-129
Return status codes,
FORTRAN coding, 2-20
MACRO coding, 2-11
obtain system messages, 4-103
summary, A-8
RMS (Record Management Services), 3-21
open file for mapping, 3-83

S
Sample programs, B-1
Schedule Wakeup ($$SCHDKW) system service, 4-131
cancel wakeups, 3-61, 4-23
example, 3-61
Search of call stack,
exception dispatcher, 3-66
Sections, 3-82
checkpoint, 3-90, 4-179
creating, 3-83, 4-46
defining extents, 3-84
deleting, 3-90
examples, 3-83, 3-86
global,
deleting, 4-70
mapping, 3-87, 4-111
mapping, 3-85, 4-46
paging, 3-89
private, 3-92, 4-46
unmapping, 3-90
using to share data, 3-89
Send Message to Accounting Manager ($$SDNACC) system service, 4-148
Send Message to Error Logger ($$NDERR) system service, 4-153
Send Message to Operator ($$NDOPR) system service, 4-154
Send Message to Symbiont Manager ($$SDSMB) system service, 4-159
Service routine,
AST, 3-12
Set AST Enable ($$AST) system service, 4-133
Set Event Flag ($$EVENT) system service, 3-6, 4-134
Set Exception Vector ($$TEXEV) system service, 3-64, 4-135
Set Power Recovery AST ($$PREVUPA) system service, 4-139
Set Priority ($$SSTPRI) system service, 4-140
Set Process Name ($$SETPRN) system service, 4-142
Set Process Swap Mode ($$SETSW) ssystem service, 4-147
example, 3-82
Set Protection on Pages ($$SETPRP) system service, 4-143
INDEX (Cont.)

Set Resource Wait Mode
($SETRWM) system service, 4-145
Set System Service Failure
Exception Mode ($SETSEM) system service, 4-146
example, 2-13
Set Timer ($SETIMR) system
service, 4-137
example with AST, 3-10
example with event flag, 3-5
examples, 3-60
Stack pointer,
modifying, 4-3
Subprocess,
deletion, 3-53
example of creating, 3-38
Suspend Process ($SUSPEND)
 system service, 4-169
Suspension, 3-47
compared with hibernation, 3-45
Swap mode,
disable, 3-82
Swapping, 3-82
disallow process swapping, 3-82
process from balance set, 3-82
Symbiont manager,
format of messages, 4-160
send message to, 4-159
Symbolic names, 2-11
obtain numeric values, A-2
page protection, A-7
processor registers, A-7
system status codes,
summary, A-9
use in error checking, 2-11
Synchronize I/O completion,
I/O status block (IOSB), 3-24
System logical name table, 3-16
System service failure
exception mode, 3-63
exception condition, 2-13
set or change, 4-146

Termination mailbox, 3-53
example, 3-54
message format, 4-39
Time,
ASCII format,
absolute time buffer, 4-15
$ASTIM, 3-57
convert to ASCII, 4-10
convert to binary, 3-57, 4-15
convert to binary integer
values, 3-61
buffer format, 4-114
system format, 3-56
obtain, 4-104
Timer and time conversion
services, 3-56
general information, 3-56
summary, 1-4
Timer requests, 3-58
cancel, 3-60, 4-22
setting, 4-135
Translate Logical Name ($STRNLOG)
 system service, 4-171
Translate,
logical name, 3-18, 4-171

Unlock Pages from Memory
($ULKPAG) system service, 4-173
Unlock Pages from Working Set
($ULWSSET) system service, 4-175
Unwind Call Stack ($UNWIND)
 system service, 4-177
example, 3-75
Unwinding the call stack, 3-74
Update Section File on Disk
($UPDSEC) system service, 4-179
User privileges, 1-1

V

VAX-11 MACRO,
coding system service calls,
$name macro, 2-3
$name_G form, 2-3
$name_S form, 2-6
Virtual address space, 3-77,
3-78, 3-79
add and delete addresses, 3-79
add pages, 4-44, 4-74
delete pages, 4-28, 4-68
layout, 3-77, 3-78
mapping sections into, 3-85
specifying arrays, 3-80

T

Table,
logical name, 3-16
Terminal driver I/O function
codes, A-3
Terminal, 3-21
assign channel, 3-21
broadcast messages to, 4-17
INDEX (Cont.)

W

Wait for Logical AND of Event Flags (SWPLAND) system service, 4-185
examples, 3-6, 3-9
Wait for Logical OR of Event Flags (SWFLOR) system service, 4-186
Wait for Single Event Flag (SWAITFR) system service, 4-182
example, 3-9
Wait,
    event flag, 3-5
I/O, 3-25

Wait (Cont.),
    resource wait mode, 2-13
    set or change, 4-145
Wake ($WAKE$) system service, 4-183
example, 3-46
Wakeup a hibernating process, 3-46
    timer scheduled, 3-61, 4-131
    cancel, 4-23
Working set, 3-81
    lock pages, 3-81
    paging, 3-81
    purge, 4-117
    size,
        changing, 3-81, 4-5
    unlock pages, 4-175

Index-9
NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

________________________________________________________________________

________________________________________________________________________

________________________________________________________________________

________________________________________________________________________

________________________________________________________________________

________________________________________________________________________

Did you find errors in this manual? If so, specify the error and the page number.

________________________________________________________________________

________________________________________________________________________

________________________________________________________________________

________________________________________________________________________

________________________________________________________________________

Please indicate the type of reader that you most nearly represent.

☐ Assembly language programmer
☐ Higher-level language programmer
☐ Occasional programmer (experienced)
☐ User with little programming experience
☐ Student programmer
☐ Other (please specify)___________________________________________________

Name________________________________________Date__________________________
Organization______________________________________________________________
Street______________________________________________________________
City________________________State__________Zip Code________________________
or Country
BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

RT/C SOFTWARE PUBLICATIONS  TW/A14
DIGITAL EQUIPMENT CORPORATION
1925 ANDOVER STREET
TEWKSBURY, MASSACHUSETTS  01876
digital