

# **P/OS System Reference Manual**

Order No. AA-N620A-TK  
Order No. AD-N620A-T1  
Order No. ADN620A-T2

October 1983

This manual describes the Professional Operating System (P/OS). It allows system and application programmers to use the operating system resources to optimize the performance of applications written for the Professional.

DEVELOPMENT SYSTEM: VAX/VMS V3.2 or later  
RSX-11M V4.1 or later  
RSX-11M-PLUS V2.1 or later  
P/OS V1.7

SOFTWARE: Professional Host Tool Kit V1.7  
PRO/Tool Kit V1.0

First Printing, December 1982

Updated, September 1983

Updated, December 1983

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.

The specifications and drawings, herein, are the property of Digital Equipment Corporation and shall not be reproduced or copied or used in whole or in part as the basis for the manufacture or sale of items without written permission.

Copyright © 1982 by Digital Equipment Corporation  
All Rights Reserved

The following are trademarks of Digital Equipment Corporation:

CTI BUS	MASSBUS	RSTS
DEC	PDP	RSX
DECmate	P/OS	Tool Kit
DECsystem-10	PRO/BASIC	UNIBUS
DECSYSTEM-20	Professional	VAX
DECUS	PRO/FMS	VMS
DECwriter	PRO/RMS	VT
DIBOL	PROSE	Work Processor
	Rainbow	

# CONTENTS

---

## **CHAPTER 1 P/OS SYSTEM OVERVIEW**

---

1.1	WHAT IS P/OS? .....	1-1
1.2	THE APPLICATION ENVIRONMENT.....	1-2
1.3	PHYSICAL, VIRTUAL, AND LOGICAL ADDRESSING.....	1-3
1.4	APPLICATION DESIGN SUGGESTIONS.....	1-3
1.4.1	Use Cooperating Tasks .....	1-3
1.4.2	Use Shared Libraries .....	1-4
1.4.3	Use Disk-Resident Overlays.....	1-4
1.4.4	Use Memory-Resident Overlays.....	1-4
1.4.5	Use Cluster Libraries .....	1-4
1.5	CHECKPOINTING.....	1-5

## **CHAPTER 2 FILE SYSTEM OVERVIEW**

---

2.1	WHAT IS RMS? .....	2-1
2.1.1	Data Storage .....	2-1
2.1.2	File Structure .....	2-3
2.1.2.1	Record Formats.....	2-3
2.1.2.2	File Organizations.....	2-3
2.1.2.3	Access Modes .....	2-4
2.2	ASSOCIATED DOCUMENTS .....	2-5

## **CHAPTER 3 USING SYSTEM DIRECTIVES**

---

3.1	DIRECTIVE PROCESSING.....	3-2
3.2	ERROR RETURNS .....	3-3
3.3	USING THE DIRECTIVE MACROS.....	3-4
3.3.1	Macro Name Conventions .....	3-6
3.3.1.1	\$ Form.....	3-6
3.3.1.2	\$C Form .....	3-7
3.3.1.3	\$S Form .....	3-7
3.3.2	DIR\$ Macro .....	3-8
3.3.3	Optional Error Routine Address.....	3-8
3.3.4	Symbolic Offsets .....	3-8
3.3.5	Examples of Macro Calls.....	3-9
3.4	FORTRAN SUBROUTINES .....	3-10
3.4.1	Using Subroutines.....	3-10
3.4.1.1	Optional Arguments.....	3-11
3.4.1.2	Task Names.....	3-11
3.4.1.3	Integer Arguments.....	3-11
3.4.1.4	GETADR Subroutine .....	3-12
3.4.2	The Subroutine Calls .....	3-12
3.4.3	Error Conditions .....	3-15
3.4.4	AST Service Routines .....	3-15
3.5	TASK STATES.....	3-16
3.5.1	Task State Transitions.....	3-17
3.6	DIRECTIVE RESTRICTIONS FOR NONPRIVILEGED TASKS .....	3-18
3.7	DIRECTIVE CATEGORIES .....	3-19
3.7.1	Task Execution Control Directives .....	3-19
3.7.2	Task Status Control Directives.....	3-20

3.7.3	Informational Directives .....	3-20
3.7.4	Event-Associated Directives .....	3-20
3.7.5	Trap-Associated Directives .....	3-22
3.7.6	I/O- and Intertask Communications-Related Directives .....	3-22
3.7.7	Memory Management Directives .....	3-22
3.7.8	Parent/Offspring Tasking Directives .....	3-22
3.8	DIRECTIVE CONVENTIONS .....	3-24

## **CHAPTER 4 LOGICAL NAMES**

---

4.1	LOGICAL NAMES AND EQUIVALENCE NAMES .....	4-1
4.1.1	The Logical Name Table .....	4-1
4.1.2	Duplicate Logical Name .....	4-2
4.2	RMS TRANSLATION OF LOGICAL NAMES .....	4-2
4.2.1	RMS and Default Directories .....	4-2
4.3	FILES-11 ACP USE OF LOGICAL NAMES .....	4-3
4.4	LOGICAL NAME CREATION .....	4-3
4.5	LOGICAL NAME TRANSLATION .....	4-3
4.6	LOGICAL NAME DELETION .....	4-3
4.7	SETTING UP A DEFAULT DIRECTORY STRING .....	4-4
4.8	RETRIEVING A DEFAULT DIRECTORY STRING .....	4-5

## **CHAPTER 5 SIGNIFICANT EVENTS, EVENT FLAGS, SYSTEM TRAPS, AND STOP-BIT SYNCHRONIZATION**

---

5.1	SIGNIFICANT EVENTS .....	5-1
5.2	EVENT FLAGS .....	5-2
5.3	SYSTEM TRAPS .....	5-4
5.3.1	Synchronous System Traps (SSTs) .....	5-4
5.3.2	SST Service Routines .....	5-5
5.3.3	Asynchronous System Traps (ASTs) .....	5-6
5.3.4	AST Service Routines .....	5-7
5.4	STOP-BIT SYNCHRONIZATION .....	5-10

## **CHAPTER 6 PARENT/OFFSPRING TASKING**

---

6.1	DIRECTIVE SUMMARY .....	6-1
6.1.1	Parent/Offspring Tasking Directives .....	6-1
6.1.2	Task Communication Directives .....	6-2
6.2	CONNECTING AND PASSING STATUS .....	6-3

## **CHAPTER 7 MEMORY MANAGEMENT DIRECTIVES**

---

7.1	ADDRESSING CAPABILITY OF A SYSTEM TASK .....	7-1
7.1.1	Address Mapping .....	7-2
7.1.2	Virtual and Logical Address Space .....	7-2
7.2	VIRTUAL ADDRESS WINDOWS .....	7-2
7.3	REGIONS .....	7-3
7.3.1	Shared Regions .....	7-7
7.3.2	Attaching to Regions .....	7-7
7.3.3	Region Protection .....	7-7
7.4	DIRECTIVE SUMMARY .....	7-8
7.4.1	Create Region Directive (CRRG\$) .....	7-8

7.4.2	Attach Region Directive (ATRG\$) .....	7-8
7.4.3	Detach Region Directive (DTRG\$) .....	7-8
7.4.4	Create Address Window Directive (CRAW\$) .....	7-8
7.4.5	Eliminate Address Window Directive (ELAW\$) .....	7-8
7.4.6	Map Address Window Directive (MAP\$) .....	7-9
7.4.7	Unmap Address Window Directive (UMAP\$) .....	7-9
7.4.8	Send By Reference Directive (SREF\$) .....	7-9
7.4.9	Receive By Reference Directive (RREF\$) .....	7-9
7.4.10	Get Mapping Context Directive (GMCX\$) .....	7-9
7.4.11	Get Region Parameters Directive (GREG\$) .....	7-9
7.5	USER DATA STRUCTURES .....	7-9
7.5.1	Region Definition Block (RDB) .....	7-10
7.5.1.1	Using Macros to Generate an RDB .....	7-12
7.5.1.2	Using Fortran to Generate an RDB .....	7-13
7.5.2	Window Definition Block (WDB) .....	7-15
7.5.2.1	Using Macros to Generate a WDB .....	7-16
7.5.2.2	Using Fortran to Generate a WDB .....	7-17
7.5.3	Assigned Values or Settings .....	7-18
7.6	PRIVILEGED TASKS .....	7-19

## **CHAPTER 8 CALLABLE SYSTEM ROUTINES**

---

8.1	GENERAL CONVENTIONS FOR ALL CALLABLE SYSTEM ROUTINES .....	8-2
8.1.1	PDP-11 R5 Calling Sequence .....	8-2
8.1.2	Conventions for Callable System Services .....	8-3
8.1.3	Status Control Block Format .....	8-3
8.2	PROATR .....	8-4
8.2.1	Status Codes Returned by PROATR .....	8-6
8.3	PRODIR .....	8-6
8.3.1	Status Codes Returned by PRODIR .....	8-7
8.4	PROFBI .....	8-7
8.4.1	Status Codes Returned by PROFBI .....	8-10
8.5	PROLOG .....	8-11
8.5.1	Creating or Translating a Logical Name .....	8-12
8.5.2	Deleting a Logical name and Set/Show .....	8-13
8.5.3	Status Codes Returned by PROLOG .....	8-15
8.6	PROTSK .....	8-16
8.6.1	Install a Task .....	8-16
8.6.2	Remove a Task, Region, or Common .....	8-18
8.6.3	Fix a Task, Region, or Common in Memory .....	8-18
8.6.4	Install/Run/Remove an Offspring Task .....	8-19
8.6.5	Status Codes Returned by PROTSK .....	8-21
8.7	PROVOL .....	8-22
8.7.1	Status Codes Returned by PROVOL .....	8-26

## **CHAPTER 9 DIRECTIVE DESCRIPTIONS**

---

9.1	FORMAT OF SYSTEM DIRECTIVE DESCRIPTIONS .....	9-1
9.1.1	ABRT\$—Abort Task .....	9-3
9.1.2	ALTP\$—Alter Priority .....	9-5
9.1.3	ALUN\$—Assign LUN .....	9-7
9.1.4	ASTX\$\$—AST Service Exit (\$S form recommended) .....	9-9
9.1.5	ATRG\$—Attach Region .....	9-12

9.1.6	CLEF\$—Clear Event Flag.....	9-14
9.1.7	CLOG\$—Create Logical Name String .....	9-17
9.1.8	CMKT\$—Cancel Mark Time Requests.....	9-17
9.1.9	CNCT\$—Connect.....	9-19
9.1.10	CRAW\$—Create Address Window .....	9-21
9.1.11	CRRG\$—Create Region.....	9-25
9.1.12	CSRQ\$—Cancel Time Based Initiation Requests.....	9-28
9.1.13	DECLS\$—Declare Significant Event (\$S Form Recommended) .....	9-30
9.1.14	DLOG\$—Delete Logical Name.....	9-31
9.1.15	DSAR\$\$ or IHAR\$\$—Disable (or Inhibit) AST Recognition (\$S Form Recommended).....	9-33
9.1.16	DSCP\$\$—Disable Checkpointing (\$S Form Recommended) .....	9-35
9.1.17	DTRG\$—Detach Region.....	9-36
9.1.18	ELAW\$—Eliminate Address Window.....	9-38
9.1.19	EMST\$—Emit Status .....	9-40
9.1.20	ENAR\$\$—Enable AST Recognition (\$S Form Recommended) .....	9-42
9.1.21	ENCP\$\$—Enable Checkpointing (\$S Form Recommended) ...	9-43
9.1.22	EXIF\$—Exit If.....	9-44
9.1.23	EXIT\$\$—Task Exit (\$S Form Recommended).....	9-46
9.1.24	EXST\$—Exit With Status .....	9-48
9.1.25	EXTK\$—Extend Task .....	9-50
9.1.26	FEAT\$—Test for specified system feature .....	9-52
9.1.27	GDIR\$—Get Default Directory .....	9-55
9.1.28	GLUN\$—Get LUN Information.....	9-57
9.1.29	GMCR\$—Get Command Line.....	9-60
9.1.30	GMCX\$—Get Mapping Context.....	9-62
9.1.31	GPRT\$—Get Partition Parameters .....	9-65
9.1.32	GREG\$—Get Region Parameters.....	9-67
9.1.33	GTIM\$—Get Time Parameters.....	9-69
9.1.34	GTSK\$—Get Task Parameters .....	9-71
9.1.35	MAP\$—Map Address Window.....	9-73
9.1.36	MRKT\$—Mark Time .....	9-76
9.1.37	QIO\$—Queue I/O Request.....	9-80
9.1.38	QIOW\$—Queue I/O Request and Wait .....	9-83
9.1.39	RCST\$—Receive Data Or Stop .....	9-85
9.1.40	RCVD\$—Receive Data .....	9-87
9.1.41	RCVX\$—Receive Data Or Exit.....	9-89
9.1.42	RDAF\$—Read All Event Flags.....	9-92
9.1.43	RDEF\$—Read Event Flag.....	9-93
9.1.44	RDXF\$—Read Extended Event Flags .....	9-94
9.1.45	RPOI\$—Request and Pass Offspring Information.....	9-96
9.1.46	RQST\$—Request Task .....	9-99
9.1.47	RREF\$—Receive By Reference.....	9-101
9.1.48	RSUM\$—Resume Task.....	9-104
9.1.49	RUN\$—Run Task .....	9-105
9.1.50	SDAT\$—Send Data .....	9-109
9.1.51	SDIR\$—Setup Default Directory String.....	9-111
9.1.52	SDRC\$—Send, Request and Connect .....	9-113
9.1.53	SDRP\$—Send Data Request and Pass Offspring Control Block.....	9-116

9.1.54	SETF\$—Set Event Flag.....	9-119
9.1.55	SFPA\$—Specify Floating Point Processor Exception AST....	9-120
9.1.56	SPND\$\$—Suspend (\$S Form Recommended).....	9-122
9.1.57	SPWN\$—Spawn .....	9-123
9.1.58	SRDA\$—Specify Receive Data AST.....	9-127
9.1.59	SREX\$—Specify Requested Exit AST Directive .....	9-129
9.1.60	SREF\$—Send By Reference .....	9-132
9.1.61	SRRA\$—Specify Receive-by-Reference AST .....	9-135
9.1.62	STIM\$—Set System Time .....	9-137
9.1.63	STLO\$—Stop For Logical OR Of Event Flags.....	9-140
9.1.64	STOP\$\$—Stop (\$S Form Recommended).....	9-142
9.1.65	STSE\$—Stop For Single Event Flag .....	9-143
9.1.66	SVDB\$—Specify SST Vector Table For Debugging Aid .....	9-144
9.1.67	SVTK\$—Specify SST Vector Table For Task .....	9-146
9.1.68	SWST\$—Switch State .....	9-148
9.1.68A	TLOG\$—Translate Logical Name .....	9-150
9.1.69	UMAP\$—Unmap Address Window.....	9-150.2
9.1.70	USTP\$—Unstop Task.....	9-152
9.1.71	VRCD\$—Variable Receive Data.....	9-153
9.1.72	VRCS\$—Variable Receive Data Or Stop .....	9-155
9.1.73	VRCX\$—Variable Receive Data Or Exit.....	9-157
9.1.74	VSDA\$—Variable Send Data .....	9-159
9.1.75	VSRC\$—Variable Send, Request and Connect.....	9-161
9.1.76	WIMP\$—What’s In My Professional.....	9-163
9.1.77	WSIG\$—Wait For Significant Event (\$S Form Recommended) .....	9-167
9.1.78	WTLO\$—Wait For Logical OR Of Event Flags .....	9-169
9.1.79	WTSE\$—Wait For Single Event Flag.....	9-171

## **CHAPTER 10    SYSTEM INPUT/OUTPUT CONVENTIONS**

---

10.1	PHYSICAL, LOGICAL, AND VIRTUAL I/O .....	10-2
10.2	SUPPORTED DEVICES.....	10-2
10.3	LOGICAL UNITS.....	10-3
10.3.1	Logical Unit Number .....	10-3
10.3.2	Logical Unit Table .....	10-3
10.3.3	Changing LUN Assignments.....	10-4
10.4	ISSUING AN I/O REQUEST .....	10-4
10.4.1	QIO Macro Format .....	10-6
10.4.2	Significant Events.....	10-8
10.4.3	System Traps .....	10-9
10.5	DIRECTIVE PARAMETER BLOCKS .....	10-10
10.6	I/O-RELATED MACROS.....	10-11
10.6.1	The QIO\$ Macro: Issuing an I/O Request.....	10-12
10.6.2	The QIOW\$ Macro: Issuing an I/O Request and Waiting for an Event Flag.....	10-13
10.6.3	The DIR\$ Macro: Executing a Directive.....	10-13
10.6.4	The .MCALL Directive: Retrieving System Macros.....	10-13
10.6.5	The ALUN\$ Macro: Assigning a LUN .....	10-14
10.6.5.1	Physical Device Names.....	10-15
10.6.5.2	Pseudo-Device Names.....	10-15
10.6.6	The GLUN\$ Macro: Retrieving LUN Information.....	10-16
10.6.7	The ASTX\$\$ Macro: Terminating AST Service.....	10-18

10.6.8	The WTSE\$ Macro: Waiting for an Event Flag .....	10-18
10.7	STANDARD I/O FUNCTIONS .....	10-19
10.7.1	IO.ATT: Attaching to an I/O Device.....	10-20
10.7.2	IO.DET: Detaching from an I/O Device.....	10-21
10.7.3	IO.KIL: Canceling I/O Requests .....	10-21
10.7.4	IO.RLB: Reading a Logical Block.....	10-22
10.7.5	IO.RVB: Reading a Virtual Block.....	10-22
10.7.6	IO.WLB: Writing a Logical Block .....	10-22
10.7.7	IO.WVB: Writing a Virtual Block .....	10-23
10.8	I/O COMPLETION .....	10-23
10.9	RETURN CODES.....	10-24
10.9.1	Directive Conditions .....	10-25
10.9.2	I/O Status Conditions.....	10-26

## **CHAPTER 11 DISK DRIVERS**

---

11.1	RX50 DESCRIPTION .....	11-1
11.2	RD50 AND RD51 DESCRIPTION.....	11-1
11.3	GET LUN INFORMATION MACRO.....	11-2
11.4	OVERVIEW OF I/O OPERATIONS.....	11-2
11.4.1	Physical I/O Operations .....	11-3
11.4.2	Logical I/O Operations.....	11-3
11.4.3	Virtual I/O Operations .....	11-4
11.5	QIO MACRO .....	11-4
11.5.1	Standard QIO Functions .....	11-4
11.6	STATUS RETURNS.....	11-6

## **CHAPTER 12 THE TERMINAL DRIVER**

---

12.1	INTRODUCTION .....	12-1
12.2	GET LUN INFORMATION MACRO.....	12-2
12.3	QIO MACRO .....	12-3
12.3.1	Subfunction Bits .....	12-4
12.3.2	Device-Specific QIO Functions.....	12-5
12.3.2.1	IO.ATA.....	12-7
12.3.2.2	IO.ATT!TF.ESQ .....	12-8
12.3.2.3	IO.CCO.....	12-8
12.3.2.4	SF.GMC.....	12-8
12.3.2.5	IO.GTS .....	12-11
12.3.2.6	IO.RAL.....	12-12
12.3.2.7	IO.RNE .....	12-13
12.3.2.8	IO.RPR .....	12-13
12.3.2.9	IO.RPR!TF.BIN.....	12-13
12.3.2.10	IO.RST.....	12-13
12.3.2.11	SF.SMC.....	12-14
12.3.2.12	IO.RTT.....	12-14
12.3.2.13	IO.WAL.....	12-15
12.3.2.14	IO.WBT.....	12-15
12.3.2.15	IO.WSD .....	12-15
12.3.2.16	IO.RSD .....	12-15
12.4	STATUS RETURNS.....	12-16
12.5	CONTROL CHARACTERS AND SPECIAL KEYS .....	12-18
12.5.1	Control Characters.....	12-18
12.5.2	INTERRUPT/DO AST Information.....	12-19



12.5.3	Special Keys.....	12-20
12.6	ESCAPE SEQUENCES.....	12-21
12.6.1	Definition.....	12-21
12.6.2	Prerequisites.....	12-22
12.6.3	Characteristics.....	12-22
12.6.4	Escape Sequence Syntax Violations.....	12-22
12.6.4.1	DEL (177).....	12-22
12.6.4.2	Control Characters (0-037).....	12-22
12.6.4.3	Full Buffer.....	12-22
12.7	VERTICAL FORMAT CONTROL.....	12-23
12.8	TYPE-AHEAD BUFFERING.....	12-24
12.9	FULL-DUPLEX OPERATION.....	12-25
12.10	INTERMEDIATE INPUT AND OUTPUT BUFFERING.....	12-25
12.11	TERMINAL-INDEPENDENT CURSOR CONTROL.....	12-25
12.12	PROGRAMMING HINTS.....	12-26

## **CHAPTER 13 THE XK COMMUNICATIONS DRIVER**

---

13.1	INTRODUCTION.....	13-1
13.2	GET LUN INFORMATION MACRO.....	13-1
13.3	QIO MACRO.....	13-2
13.3.1	Device-Specific QIO Functions.....	13-4
13.3.1.1	IO.ANS.....	13-4
13.3.1.2	IO.ATA.....	13-4
13.3.1.3	IO.BRK.....	13-4
13.3.1.4	IO.CON.....	13-4
13.3.1.5	SF.GMC.....	13-5
13.3.1.6	IO.HNG.....	13-8.1
13.3.1.7	IO.LTI.....	13-8.1
13.3.1.8	IO.ORG.....	13-9
13.3.1.9	IO.RAL.....	13-9
13.3.1.10	IO.RNE.....	13-9
13.3.1.11	SF.SMC.....	13-9
13.3.1.12	IO.TRM.....	13-9
13.3.1.13	IO.UTI.....	13-9
13.3.1.14	IO.WAL.....	13-9
13.4	STATUS RETURNS.....	13-10
13.5	FULL-DUPLEX OPERATION.....	13-11
13.6	UNSOLICITED EVENT PROCESSING.....	13-11
13.6.1	XTU.UI.....	13-11
13.7	TIME-OUT.....	13-11
13.7.1	Read requests.....	13-12
13.7.2	IO.CON.....	13-12
13.7.3	IO.ORG.....	13-12
13.8	XON/XOFF SUPPORT.....	13-12

## **APPENDIX A STANDARD ERROR CODES**

---

## **APPENDIX B SUMMARY OF I/O FUNCTIONS**

---

B.1	DISK DRIVER.....	B-1
B.2	TERMINAL DRIVER.....	B-2
B.2.1	Subfunction Bits for Terminal-Driver Functions.....	B-2

## **APPENDIX C I/O FUNCTION AND STATUS CODES**

---

C.1	I/O STATUS CODES .....	C-1
C.1.1	I/O Status Error Codes .....	C-2
C.1.2	I/O Status Success Codes .....	C-3
C.2	DIRECTIVE CODES .....	C-4
C.2.1	Directive Error Codes .....	C-4
C.2.2	Directive Success Codes .....	C-4
C.3	I/O FUNCTION CODES .....	C-4
C.3.1	Standard I/O Function Codes .....	C-4
C.3.2	Specific Terminal I/O Function Codes .....	C-5
C.3.3	Subfunction Bits .....	C-6

## **APPENDIX D FACILITY AND ERROR CODES**

---

D.1	SUB-FACILITY CODES .....	D-1
D.2	FATAL ERROR CODES .....	D-2
D.3	BUGCHECK .....	D-2

## **INDEX**

### **FIGURES**

---

3-1	Directive Parameter Block (DPB) Pointer on the Stack .....	3-4
3-2	Directive Parameter Block (DPB) on the Stack .....	3-5
7-1	Virtual Address Windows .....	7-4
7-2	Region Definition Block .....	7-5
7-3	Mapping Windows to Regions .....	7-6
7-4	Region Definition Block .....	7-11
7-5	Window Definition Block .....	7-14
10-1	QIO Directive Parameter Block .....	10-11

### **TABLES**

---

3-1	Fortran Subroutines and Corresponding Macro Calls .....	3-13
3-2	Directives Not Available as Subroutines .....	3-15
3-3	System Directives that can be Issued by Nonprivileged Tasks .....	3-19
3-4	Task Execution Control Directives .....	3-20
3-5	Task Status Control Directives .....	3-21
3-6	Informational Directives .....	3-21
3-7	Event Associated Directives .....	3-21
3-8	Trap Associated Directives .....	3-22
3-9	I/O- and Intertask Communications Related Directives .....	3-23
3-10	Memory Management Directives .....	3-23
3-11	Parent/offspring Tasking Directives .....	3-24
5-1	Trap Vector Table .....	5-5
6-1	Directive Examples For Intertask Synchronization .....	6-4
7-1	Bits of the Region Status Word .....	7-11
7-2	RDB Array Format .....	7-14
7-3	WDB Format .....	7-15
7-4	WDB Array Format .....	7-18
8-1	Accessible File Attributes .....	8-5
8-2	PROFBI Status Codes .....	8-10
8-3	PROLOG status Codes .....	8-16

8-4	PROTSK Status Codes .....	8-21
8-5	PROVOL Status Codes .....	8-26
9-1	Region Definition Block Parameters .....	9-13
9-2	Window Definition Block Parameters .....	9-22
9-3	Region Definition Block Parameters .....	9-26
9-4	Region Definition Block Parameters .....	9-37
9-5	Window Definition Block Parameters .....	9-38
9-6	System Feature Symbols .....	9-53
9-7	Window Definition Block Parameters .....	9-63
9-8	Window Definition Block Parameters .....	9-74
9-9	Window Definition Block Parameters .....	9-102
9-10	Window Definition Block Parameters .....	9-133
9-11	Window Definition Block Parameters .....	9-150.2
9-12	The Configuration Table Output Buffer Format .....	9-165
10-1	Physical Device Names .....	10-15
10-2	Pseudo Device Names .....	10-15
10-3	Get LUN Information .....	10-17
10-4	Binary Status Codes .....	10-25
10-5	Directive Conditions .....	10-25
10-6	I/O Status Conditions .....	10-27
11-1	Standard Disk Devices .....	11-1
11-2	Buffer Get LUN Information for Disks .....	11-2
11-3	Standard QIO Functions for Disks .....	11-4
11-4	Disk Status Returns .....	11-6
12-1	Buffer Get LUN Information For Terminals .....	12-2
12-2	Standard and Device-Specific QIO Functions for Terminals .....	12-3
12-3	Definition of Subfunction Bit .....	12-5
12-4	Summary of Subfunction Bits .....	12-6
12-5	Driver-Terminal Characteristics for SF.GMC and SF.SMC Functions .....	12-9
12-6	TC.TTP (Terminal Type) Values Set by SF.SMC and Returned by SF.GMC .....	12-10
12-7	Receiver and Transmitter Speed Values (TC.RSP, TC.XSP) .....	12-11
12-8	Information Returned by Get Terminal Support (IO.GTS) QIO .....	12-12
12-9	Terminal Status Returns .....	12-17
12-10	Terminal Control Characters .....	12-19
12-11	Special Terminal Keys .....	12-21
12-12	Vertical Format Control Characters .....	12-23
13-1	Buffer Get LUN Information for XK Driver .....	13-2
13-2	Standard and Device Specific QIO Functions .....	13-2
13-3	XK Driver Characteristics for SF.GMC and SF.SMC Functions .....	13-5
13-4	TC.FSZ and TC.PAR Relationship .....	13-6
13-5	Receiver and Transmitter Speed Values (TC.RSP, TC.XSP) .....	13-7
13-6	XK Driver Status Returns .....	13-10
13-7	Unsolicited Event Types .....	13-11



# PREFACE

---

## **MANUAL OBJECTIVES AND INTENDED AUDIENCE**

The *P/OS System Reference Manual* describes the base system software supporting the Professional 300 Series personal computer. This manual is for experienced system programmers and applications programmers who use the P/OS (the Professional Operating System) system resources to optimize the performance of applications programs written for the Professional. This manual is especially helpful for programmers who have experience with RSX-11M-PLUS systems. Applications programmers using high-level languages (such as PRO/BASIC-PLUS-2) may also find this manual useful.

## **STRUCTURE OF THIS DOCUMENT**

Chapter 1 is an overview of the P/OS system. It contrasts P/OS features with RSX-11M-PLUS features (on which P/OS has been based). The chapter also provides applications design suggestions.

Chapters 2 through 7 describe the types of system directives, logical names, and task execution control mechanisms.

Chapter 8 defines the callable system utilities.

Chapter 9 describes all of the system directives in detail.

Chapter 10 is a detailed discussion of input and output conventions.

Chapter 11 describes the P/OS disk drivers (device handlers).

Chapter 12 describes the P/OS terminal driver.

Chapter 13 describes the P/OS communications driver.

The four appendixes cover system error messages and I/O function and status codes.

## **ASSOCIATED DOCUMENTS**

Please refer to the other manuals in the Tool Kit Documentation Set for more information on developing applications for the Professional.

## CONVENTIONS USED IN THIS DOCUMENT

The following conventions apply in this manual:

<i>Convention</i>	<i>Meaning</i>
UPPERCASE WORDS AND LETTERS	Uppercase words and letters, used in examples, indicate that you should type the word or letter exactly as shown.
lowercase words	Lowercase words and letters, used in examples, indicate that you are to substitute a word or value.
[optional]	Square brackets indicate optional entries in a command line. Note that when an option is entered, the brackets are not included in the command line. Square brackets also are a part of the User File Directory (UFD) and User Identification Code (UIC) syntax. When you use a UFD or UIC (in a file specification, for example), brackets are required syntax elements; that is, they do not indicate optional entries.
...	A horizontal ellipsis indicates that the preceding item can be repeated one or more times. For example:  file-spec[,file-spec...]
.	A vertical ellipsis indicates that not all of the statements in an example are shown.

# CHAPTER 1

## P/OS SYSTEM OVERVIEW

---

This chapter provides a brief overview of P/OS, contrasts P/OS with RSX-11M-PLUS, and provides application design suggestions.

### 1.1 WHAT IS P/OS?

P/OS is a multitasking, real-time, resource-sharing operating system. It is based on the RSX-11M-PLUS operating system. Some RSX-11M-PLUS software features remain the same on P/OS, some have been removed, some have changed, and some new software features have been added. The principal difference between P/OS and RSX is that the normal RSX user interface has been replaced by a menu system. Furthermore, some of the RSX utilities carried over to P/OS are now program callable routines.

#### **Summary of Differences**

##### **RSX-11M-PLUS features not available on P/OS:**

- Group global event flags
- Virtual terminals and batch processing
- Alternate CLI support
- External task headers
- MCR and LOAD
- VMR
- Indirect Command Processor
- TDX (catchall)
- HELLO, BYE, ACNT
- Console Logging

- Error logging
- System accounting
- Shadow recording
- Queue Manager (QMG)
- File Control Services (FCS)
- Disk swapping
- System generation
- System reconfiguration (CON and HRC)
- Checkpointing for common regions (CPRC\$)
- Prototype Task Control Blocks in secondary pool

**RSX-11M-PLUS features modified for P/OS:**

- Terminal driver
- SAVE
- System utilities (FMT, BAD, INI, INSTALL, FIX, REMOVE, UFD)
- GET TIME

**New P/OS features:**

- Logical name directives
- Segmented libraries
- RMS V2.0
- Switch State directive (SWST\$)
- Automatic volume mounting and dismounting
- Enhanced higher-level language interface to the system and the utilities (POSSUM library).

## 1.2 THE APPLICATION ENVIRONMENT

A healthy understanding of how the operating system works can help you write applications programs that maximize the system's resources. A thorough understanding of the system takes time and experience. However, the following sections discuss aspects of the system that will help you begin to understand it.

To fully understand the information provided in the following sections, you should be familiar with the terminology used here (such as common regions and task regions). Please refer to the *RSX-11M/M-PLUS Task Builder Manual* for detailed explanations of these terms.



### 1.3 PHYSICAL, VIRTUAL, AND LOGICAL ADDRESSING

The primary addressing mechanism of the Professional is the 16-bit computer word. This means that the maximum amount of physical memory that a task may access at a single point in time is limited to 32K words. However, the presence of hardware memory management enables a task, using the P/OS memory management (PLAS) directives, to access more than 32K words.

Physical addresses are single locations in memory. Virtual addresses are the addresses within a task. Logical addresses are the actual physical memory addresses that the task can access. Physical and virtual address spaces are contiguous. However, a task's logical address space need not be contiguous in physical memory. (See Chapter 2 of the *RSX-11M/M-PLUS Task Builder Manual* for a complete discussion of addressing concepts.)

Using P/OS system features to manipulate logical address space allows you to make use of more than 32K words of virtual address space. Furthermore, the multitasking capabilities of P/OS allow you to design applications that can consist of multiple, cooperating, concurrent tasks.

### 1.4 APPLICATION DESIGN SUGGESTIONS

The following sections list suggestions for designing applications that make the most efficient use of the P/OS multitasking, resource-sharing capabilities. In particular, these suggestions may help you to design programs that might otherwise exceed the 32K word virtual address space limitation of a task.

#### 1.4.1 Use Cooperating Tasks

An application is a task or set of tasks that perform a needed function or set of functions. The application may consist of multiple, cooperating tasks that pass context (variables) between tasks by using data packets, command lines, and shared memory. A task may be requested using the following system directives:

- SPWN\$*—useful when passing a command line and there is a need to receive status from the cooperating task.
- RPOI\$*—useful when passing a command line and there is no need to receive status from the cooperating task.
- SDRC\$ and VSRC\$*—useful when passing data packets and there is a need to receive status from the cooperating task.
- RQST\$*—useful when simply requesting a task and there is no need to receive status from the cooperating task.

Additional context may be passed using the *SDAT\$*, *VSDA\$*, and *SREF\$*. See Chapters 6 and 9 for more details on using these directives.

### 1.4.2 Use Shared Libraries

A shared library is a block of code that resides in memory and can be used by any number of tasks. Since the library routines are available to any task, you save physical memory by having only one copy of these routines in memory rather than duplicating them in each task. Furthermore, when the library is not being accessed (mapped) by any task, the system writes the library out to disk and removes it from memory to make room for other tasks as necessary. (If the library is read-only, the system removes the library region from memory but does not write it out to disk.) Chapter 5 of the *RSX-11M/M-PLUS Task Builder Manual* discusses shared libraries.

### 1.4.3 Use Disk-Resident Overlays

You can divide an application task into pieces called segments. Several segments of a task share a given section of the task's virtual address space, but only one segment may be in memory at one time. Segments are individually read from the disk into a section of the task's address space as needed, overwriting a previously read segment. A task constructed of disk-resident overlays reduces the memory and virtual address space needed by the task. Chapters 3 and 4 of the *RSX-11M/M-PLUS Task Builder Manual* discuss segments and disk-resident overlays in detail.

### 1.4.4 Use Memory-Resident Overlays

Memory-resident overlays are different from disk-resident overlays in that all of the task's segments are present in physical memory at the same time. A segment is mapped into a section of the task's virtual address space as needed by using memory management directives. As it maps each new segment, the task's logical address space changes as it maps each new segment into the task's virtual address space and unmaps the previous segment.

A task constructed of memory-resident overlays reduces the virtual address space needed by the task but does not reduce the physical memory requirements. However, tasks constructed of memory-resident overlays are faster since they do not involve disk I/O. Chapters 3 and 4 of the *RSX-11M/M-PLUS Task Builder Manual* discuss memory-resident overlays in detail.

### 1.4.5 Use Cluster Libraries

A cluster library is both a function and a structure that allows tasks to dynamically map memory-resident, shared libraries at run time. The advantage of using cluster libraries is that they save task virtual address space by using the same section of task virtual address space to map independent memory-resident, shared libraries. Chapter 5 of the *RSX-11M/M-PLUS Task Builder Manual* describes cluster libraries at length.

## 1.5 CHECKPOINTING

Checkpointing is the process of writing a task or common to a file on a disk to make room for a higher priority task or common competing for memory. Given that a task or common is capable of being checkpointed, tasks and commons compete for memory based on their respective priorities. (The priority value of a common region is equal to one greater than the highest priority task mapped to that common region.)

Two types of task states affect the possibility that a task or common can be checkpointed. The first type prevents a checkpoint from occurring at all. The second type enhances the possibility that a task or common will be checkpointed.

The following conditions prevent a checkpoint from occurring:

- A noncheckpointable region (specified at task-build time)
- A task region with checkpointing disabled (DSCP\$)
- An exiting task
- A region with resident, mapped tasks—that is, all currently mapped tasks must be checkpointed before the region itself is eligible for checkpointing
- A region with outstanding I/O

The following conditions increase the possibility of a task or region being checkpointed:

- A stopped task has an effective memory priority of zero
- A checkpointable task doing synchronous terminal I/O (since the task's terminal I/O is buffered and the task is stopped until the I/O completes)
- A task which previously had checkpointing disabled can issue the Enable Checkpointing directive (ENCP\$)



## CHAPTER 2

# FILE SYSTEM OVERVIEW

---

This chapter is an overview of the file system supported on P/OS. It is intended as an introduction to the Record Management System (RMS).

### 2.1 WHAT IS RMS?

RMS (Record Management Services) is a set of routines that allows programs to store, retrieve, and process (modify and delete) records and files. RMS provides the connection between a program and the stored data the program requires.

The ability to store and retrieve information and the ability to process that information readily and in an orderly fashion requires that the information be stored in an orderly fashion, such as records. A record is a logical unit of data; that is, an item or collection of related items.

To keep the records of one type separate from records of another type, records are organized into files. A file contains groups of records of the same type. One or more files, depending on the amount of data, contain all the records of a specific type.

How the data is used helps determine how the records are stored in and retrieved from files—that is, access.

#### 2.1.1 Data Storage

The data that your programs use is typically stored on mass storage devices called disks. P/OS supports both a hard disk and flexible diskettes. The operating system software controls the disk devices, and allows your programs to access the data stored on them. Each device is governed by a device driver—the software that handles the I/O.

The Files-11 Ancillary Control Processor (FCP) is the software that catalogues and maintains files on the disks and makes I/O requests to the device drivers.

The smallest unit of information stored on a disk is a bit. A bit is an area of disk surface for which the magnetic orientation can be changed to one of two values, conventionally designated 0 and 1.

Information is usually grouped into units of 8 bits, called bytes. Bytes are used, for example, to represent alphanumeric characters in memory with the DEC Multinational Character Set. Other ways of representing data, particularly numeric data, may require 2 or more bytes. A word, for example, consists of 2 bytes (or 16 bits).

Data is stored hierarchically on a disk, as follows. A sector consists of 512 8-bit bytes. A track consists of all the sectors at a single radius on one disk platter, and a cylinder consists of all the tracks at the same radius on all the platters. The disk drive can access all tracks on a single cylinder without changing position, which affects speed of data access.

The FCP imposes a logical structure on each disk. It treats the disk as a single, logically contiguous, series of data units, called blocks. A block contains 512 8-bit bytes. Logical blocks are numbered sequentially, from 0 to n-1, where n is the number of blocks on the disk.

On disk, a file is simply a series of blocks, which contain your data organized into records. The FCP treats each file as a device, ignoring any blocks on the disk except those in the file being processed.

The blocks in a file, however, need not be logically contiguous. As files are created or extended, the file processor may allocate blocks to the file that are not next to each other on the disk. The blocks in a file, then, are virtually contiguous. Virtual blocks are numbered sequentially in a file from 1 to n, where n is the last block in a file.

Note that a Virtual Block Number (VBN) and a Logical Block Number (LBN) refer to the same physical unit of disk storage space. But although a virtual block also has an LBN, a logical block has a VBN only if it is allocated to a file.

To access files, the system translates VBNs to LBNs and makes an I/O request to the device driver. The device driver, in turn, translates the LBNs to the physical location (cylinder, track, and sector) that is to be read or written.

Disk storage allows random access. Also called direct access, this means that a specific record can be located and retrieved without a search of all the records that precede it in the file. The time needed to access a record may therefore be improved.

In addition, disk storage allows access sharing. This means that more than one task can access a disk at a time, and more than one task can be allowed to open the same file at one time.

## 2.1.2 File Structure

The operating system software (that is, the file processor, and device drivers) handle files. Your programs, however, must be able to access the records within the files so they can process the data within the records.

RMS allows you to define the internal structure of files (the size and arrangement of records within files) and provides operations that allow your programs to read and write records in files. RMS thus provides the interface between the operating system and your programs.

You define the internal structure of a file when you create it by selecting:

- Record format
- File organization
- Access modes

**2.1.2.1 Record Formats** —RMS does not handle, or process, data within records. Your program does that. However, to retrieve and store records for your program, RMS must know how large that record is. Five record formats, therefore, are available so you can define for RMS what size your data records are:

- Fixed length*—every record is the same size.
- Variable length*—records can be of different lengths up to a maximum size that you specify.
- Variable length with fixed control (VFC)*—a fixed-length control area precedes a variable-length data area in each record.
- Stream*—a record consists of a continuous series of DEC multinational characters delimited by a special character called a terminator.
- Undefined*—a file with undefined records may have either no record format or may contain records that are not in one of the four formats just described.

RMS's support of stream and undefined records provides compatibility with non-RMS files or other DEC systems.

**2.1.2.2 File Organizations** —The arrangement of records in files directly affects how quickly and easily RMS can access those records. Your selection of file organization, therefore, should take access mode into consideration. (The next section introduces access modes.)

RMS makes three file organizations available:

- Sequential*—in a sequential file, records are arranged within the file in the order in which they were written. You can add records to and delete records from a sequential file only at the physical end of the file.

- *Relative*—a relative file consists of a series of cells of a fixed size. The cells are numbered consecutively from 1 to n, where n is the number of cells in the file. The cell numbers are known as relative record numbers (RRN).

Each record in the file is stored in a cell and is accessed by the cell's relative record number. Because records are stored by relative record number, they may not be stored in the order that they are written.

- *Indexed*—in an indexed file, records are arranged in ascending order by key. A key is a data field within the record that RMS uses to determine the order in which to access the records. This allows a record to be identified by its contents, not by its position.

When you create an indexed file, you must define one field of the record as the primary key. A key is defined by its location within the record and its length. When a record is stored in that file, RMS inserts the record in order by the value that is to be stored in the primary key field – that is, after a record with a lower or equal value in the primary key field and before a record with a higher value in the primary key field.

You can optionally define other record fields as alternate keys. These keys specify alternate access orders for the retrieved records.

For each field defined as a primary or alternate key, RMS constructs an index. A primary index contains the values in the primary key fields, the first alternate index contains the values in the first alternate key field, and so on. Each index entry points to the one data record associated with that key value.

Thus, each key value provides a logical access path to locate a specific record or set of records within a file. The indexes also allow your program to retrieve the records in a specific order. RMS stores the indexes in the file itself.

**2.1.2.3 Access Modes** —The access modes are the methods that RMS-11 uses to store and retrieve the contents of files. The contents of files can consist of either records or blocks.

Record access modes

- For sequential access, record storage and retrieval begins at a point in the file and continues with consecutive records through the file. Your program issues a series of requests to RMS-11 to successively retrieve the next record in the file.
- For RFA access, RMS-11 uses the record file address (RFA) as an identifier to gain direct access to a specific record in a file, without a successive search of all the records that precede it. The RFA is a unique record identifier that RMS-11 establishes for every record that it writes to a disk file. The RFA remains valid for the record. If a record is deleted, its RFA is not reused.

When RMS-11 stores a record in a file, it establishes the RFA for that record and returns the RFA information to your program. Your program can then use the RFA to retrieve the record.



Note that because only RMS-11 can establish the RFA, you cannot store a record by RFA (that is, specify an RFA for the record).

- For key access, your program specifies an identifier that allows RMS-11 to gain direct access to a specific record, without a successive search of all the records that precede it. For sequential files with fixed-record format or for relative files, this identifier is a relative record number (RRN). For indexed files, this identifier is a key value.

If the identifier is a relative record number, RMS-11 stores or retrieves the record in that cell in a relative file, or in that position in a sequential file.

If the identifier is a key value, RMS-11 stores or retrieves the record associated with that key value in an indexed file. You can also specify that all records are to be retrieved in order by primary or alternate key value.

#### Block access modes

- For sequential access, RMS-11 stores and retrieves data as a consecutive series of 512-byte blocks. Your program issues a series of requests to successively store or retrieve the next block in the file. This means that RMS-11 can process not only files of any RMS-11 organization but non-RMS-11 files as well.
- For VBN access, your program specifies the virtual block number (VBN) as the identifier of the block to be accessed. RMS-11 uses the block number to gain direct access to the specified block, without a search of all the blocks that precede it.

## 2.2 ASSOCIATED DOCUMENTS

More detailed information on using RMS is available in the the following RMS manuals:

*RSX-11M/M-PLUS RMS-11: An Introduction* presents the major concepts of RMS-11, including record formats, file organizations, and record access modes.

*The RSX-11M/M-PLUS RMS-11 User's Guide* provides detailed information for both MACRO-11 and high-level language programmers on file and task design using RMS-11.

*The RSX-11M/M-PLUS RMS-11 Macros and Symbols* manual is a reference document for MACRO-11 programmers describing the macros and symbols that make up the interface between a MACRO-11 program and the RMS-11 operation routines.

In addition, the *RSX-11M/M-PLUS RMS-11 Mini-Reference Insert* is a quick-reference guide for users who are familiar with RMS-11 and its documentation. Also, two other manuals are available which are *PRO/RMS-11: An Introduction* and *PRO/RMS-11 Macro Programmer's Guide*.



## CHAPTER 3

# USING SYSTEM DIRECTIVES

---

When a task requires the Executive to perform an operation, the task issues a system directive to make the request. System directives allow you to control the execution and interaction of tasks. If you are a MACRO-11 programmer, you usually issue directives in the form of macros defined in the system macro library. If you are a Fortran programmer, you must issue system directives in the form of calls to subroutines contained in the system object module library. These are the libraries provided in the Tool Kit. Programs written in other higher-level languages that provide support for the PDP-11 standard R5 calling conventions for Fortran may also make use of these calls (see Section 8.1.1). Check your language reference manual and user's guide to determine if you are using that format.

System directives enable tasks to:

- Obtain task and system information
- Measure time intervals
- Perform I/O functions
- Spawn other tasks
- Communicate and synchronize with other tasks
- Manipulate a task's logical and virtual address space
- Suspend and resume execution
- Exit

Directives are implemented by the EMT 377 instruction. EMT 0 through EMT 376 are considered to be non-RSX EMT synchronous system traps. They cause the Executive to abort the task unless the task has specified that it wants to receive control when such traps occur.

If you are a MACRO-11 programmer, use the system directive macros supplied in the system macro library for directive calls, rather than hand-coding calls to directives. Then you need only reassemble your program to incorporate any changes in the directive specifications.

Sections 3.1, 3.2, and 3.5 are intended for all users. Section 3.3 specifically describes the use of macros, while Section 3.4 describes the use of Fortran subroutine calls. Programmers using other supported languages should refer to the appropriate language reference manual supplied by DIGITAL.

### 3.1 DIRECTIVE PROCESSING

Processing a system directive involves four steps:

1. A user task issues a directive with arguments that are used only by the Executive. The directive code and parameters that the task supplies to the system are known as the Directive Parameter Block (DPB). The DPB can be either on the user task's stack or in a user task's data section.
2. The Executive receives an EMT 377 generated by the directive macro (or a DIR\$ macro) or Fortran subroutine.
3. The Executive processes the directive.
4. The Executive returns directive status information to the task's Directive Status Word (DSW).

Note that the Executive preserves all task registers when a task issues a directive.

The user task issues an EMT 377 (generated by the directive) together with the address of a DPB or a DPB itself, on the top of the user task's stack. When the stack contains a DPB address, the Executive removes the address after processing the directive, and the DPB itself remains unchanged. When the stack contains the actual DPB rather than a DPB address, the Executive removes the DPB from the stack after processing the directive.

The first word of each DPB contains a Directive Identification Code (DIC) byte and a DPB size byte. The DIC indicates which directive is to be performed; the size byte indicates the DPB length in words. The DIC is in the low-order byte of the word, and the size is in the high-order byte.

The DIC is always an odd-numbered value. This allows the Executive to determine whether the word on the top of the stack (before EMT 377 was issued) was the address of the DPB (even-numbered value) or the first word of the DPB (odd-numbered value).

The Executive normally returns control to the instruction following the EMT. Exceptions to this are directives that result in an exit from the task that issued them and an Asynchronous System Trap (AST) exit.

The Executive also clears or sets the Carry bit in the Processor Status Word (PSW) to indicate acceptance or rejection, respectively, of the directive. The DSW, addressed symbolically as \$DSW<sup>1</sup>, is set to indicate a more specific cause for acceptance or rejection of the directive. The DSW usually has a value of +1 for acceptance and a range of negative values for rejection. (Exceptions to this rule are multiple success return codes for directives such as CLEF\$, SETF\$, and GPRT\$, among others). The Executive associates DSW values with symbols, using mnemonics that report either successful completion or the cause of an error (see Section 3.2). The Instrument Society of America (ISA) Fortran calls, CALL START and CALL WAIT, are exceptions, since ISA requires positive numeric error codes. See Sections 9.1.36 and 9.1.49 for details; the specific return values are listed there with each directive.

In the case of successful Exit directives, the Executive does not, of course, return control to the task. If an Exit directive fails, however, control returns to the task with an error status in the DSW.

On Exit, the Executive frees task resources as follows:

- Detaches all attached devices.
- Flushes the AST queue and despecifies all specified ASTs.
- Flushes the receive and receive-by-reference queues.
- Flushes the clock queue for outstanding Mark Time requests for the task.
- Closes all open files (files open for write access are locked).
- Detaches all attached regions except in the case of a fixed task, where no detaching occurs.<sup>1</sup>
- Runs down the task's I/O.
- Disconnects from interrupts.
- Breaks the connection with any offspring tasks.
- Frees the task's memory if the task was not fixed.

If the Executive rejects a directive, it usually does not clear or set any specified event flag. Thus, the task may wait indefinitely if it indiscriminately executes a Wait For directive corresponding to a previously issued Mark Time directive that the Executive has rejected. You should always ensure that a directive has been completed successfully.

### 3.2 ERROR RETURNS

As stated earlier, the Executive associates the error codes with mnemonics that report the cause of the error. In the text of this manual, the mnemonics are used exclusively. The macro DRERR\$, which is expanded in Appendix A, provides a correspondence between each mnemonic and its numeric value.

---

1. The Task Builder resolves the address of \$DSW.

Appendix A also gives the meaning of each error code. In addition, each directive description in Chapter 9 contains specific, directive-related interpretations of the error codes.

### 3.3 USING THE DIRECTIVE MACROS

If you are programming in MACRO-11, you must decide how to create the DPB before you issue a directive. The DPB may either be created on the stack at run time (see Section 3.3.1.3, which describes the \$S form of directive) or created in a data section at assembly time (see Sections 3.3.1.1 and 3.3.1.2, which describe the \$ form and \$C form, respectively). If parameters vary and the code must be reentrant, the DPB must be created on the stack.

Figures 3-1 and 3-2 illustrate the alternative directives and also show the relationship between the stack pointer and the DPB.

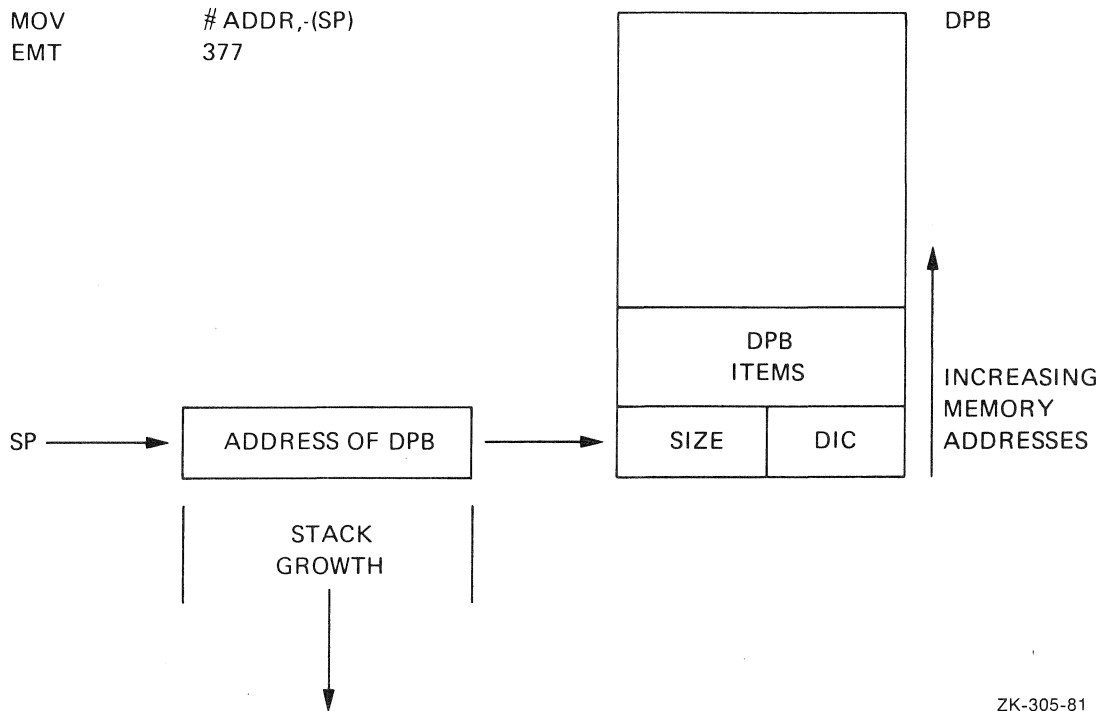
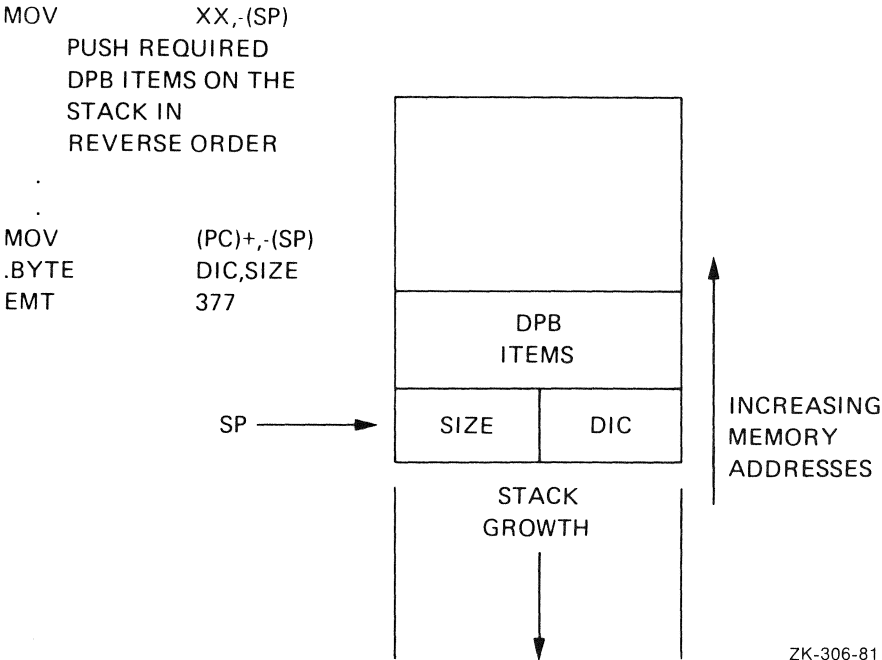


Figure 3-1  
Directive Parameter Block (DPB) Pointer on the Stack



ZK-306-81

Figure 3-2  
Directive Parameter Block (DPB) on the Stack

### 3.3.1 Macro Name Conventions

When you are programming in MACRO-11, you use system directives by including directive macro calls in your programs. The macros for the system directives are contained in the System Macro Library (RSXMAC.SML). The .MCALL assembler directive makes these macros available to a program. The .MCALL arguments are the names of all the macros used in the program. For example:

```

;
; CALLING DIRECTIVES FROM THE SYSTEM MACRO LIBRARY
; AND ISSUING THEM.
;
.MCALL MRKT$S,WTSE$S
.
.
.
Additional .MCALLs or code
.
.
.
MRKT$S #1,#1,#2,,ERR ;MARK TIME FOR 1 SECOND
WTSE$S #1 ;WAIT FOR MARK TIME TO COMPLETE
.
.
.

```

Macro names consist of as many as four letters, followed by a dollar sign (\$) and, optionally, a C or an S. The optional letter or its absence specifies which of three possible macro expansions the programmer wants to use.

**3.3.1.1 \$ Form** —The \$ form is useful for a directive operation that is to be issued several times from different locations in a non-reentrant program segment. The \$ form is most useful when the directive is issued several times with varying parameters (one or more but not all parameters change), or in a reentrant program section when a directive is issued several times even though the DPB is not modified. The \$ form produces only the directive's DPB and must be issued from a data section of the program. The code for actually executing a directive that is in the \$ form is produced by a special macro, DIR\$ (discussed in Section 3.3.2).

Because execution of the directive is separate from the creation of the directive's DPB:

1. A \$ form of a given directive needs to be issued only once (to produce its DPB).
2. A DIR\$ macro associated with a given directive can be issued several times without incurring the cost of generating a DPB each time it is issued.
3. The directive's parameters can be easily accessed and changed by labeling the start of the DPB and using the offsets defined by the directive.



When a program issues the \$ form of macro call, the parameters required for DPB construction must be valid expressions for MACRO-11 data storage instructions (such as .BYTE, .WORD, and .RAD50). You can alter individual parameters in the DPB. You might do this if you want to use the directive many times with varying parameters.

**3.3.1.2 \$C Form** —Use the \$C form when a directive is to be issued only once. The \$C form eliminates the need to push the DPB (created at assembly time) onto the stack at run time. Other parts of the program, however, cannot access the DPB because the DPB address is unknown. Note, in the \$C form of the macro expansion (see Section 3.3.5), that the new value of the assembler's location counter redefines the DPB address \$\$\$ each time an additional \$C directive is issued.

The \$C form generates a DPB in a separate p-section<sup>2</sup> called \$DPB\$. The DPB is first followed by a return to the user-specified p-section, then by an instruction to push the DPB address onto the stack, and finally by an EMT 377. To ensure that the program reenters the correct p-section, you must specify the p-section name in the argument list immediately following the DPB parameters. If the argument is not specified, the program reenters the blank (unnamed) p-section.

The \$C form also accepts an optional final argument that specifies the address of a routine to be called (by a JSR instruction) if an error occurs during the execution of the directive (see Section 3.3.2).

When a program issues the \$ form of a macro call, the parameters required for DPB construction must be valid expressions for MACRO-11 data storage instructions (such as .BYTE, .WORD, and .RAD50). (This is not true for the p-section argument and the error routine argument, which are not part of the DPB.)

**3.3.1.3 \$\$ Form** —Program segments that need to be reentrant and use DPBs with dynamic parameters should use the \$\$ form. Only the \$\$ form produces the DPB at run time. The other two forms produce the DPB at assembly time.

In this form, the macro produces code to push a DPB onto the stack, followed by an EMT 377. In this case, the parameters must be valid source operands for MOV-type instructions. For a 2-word Radix-50 name parameter, the argument must be the address of a 2-word block of memory containing the name. Note that you should not use the Stack Pointer (or any reference to the Stack Pointer) to address directive parameters when the \$\$ form is used.<sup>3</sup> (In the example in Section 3.3.1, the error routine argument ERR is a target address for a JSR instruction; see Section 3.3.3.)

---

2. Refer to the *PDP-11 Language Reference Manual* for a description of p-sections (program sections).

3. Subroutine or macro calls can use the stack for temporary storage, thereby destroying the positional relationship between SP and the parameters.

Note that in the \$\$ form of the macro, the macro arguments are processed from right to left. Therefore, when using code of the form:

```
MACRO $$ , , (R4)+ , (R4)+
```

the result may be obscure.

### 3.3.2 DIR\$ Macro

The DIR\$ macro allows you to execute a directive with a DPB previously defined by the \$ form of a directive macro. This macro pushes the DPB address onto the stack and issues an EMT 377 instruction.

The DIR\$ macro generates an Executive trap using a previously defined DPB:

Macro Call: DIR\$ adr,err

Note: adr and err are optional.

adr	The address of the DPB. (The address, if specified, must be a valid source address for a MOV instruction.) If this address is not specified, the DPB or its address must be on the stack.
err	The address of the error return (see Section 3.3.3). If this error return is not specified, an error simply sets the Carry bit in the Processor Status Word.

Note: DIR\$ is not an Executive directive and does not behave as one. There are no variations in the spelling of this macro.

### 3.3.3 Optional Error Routine Address

The \$C and \$\$ forms of macro calls and the DIR\$ macro can accept an optional final argument; note that the DIR\$ macro is not an Executive directive (DIR\$C and DIR\$\$ are not valid macro calls). The argument must be a valid assembler destination operand that specifies the address of a user error routine. For example, the DIR\$ macro

```
DIR$      #DPB,ERROR
```

generates the following code:

```
MOV      #DPB, -(SP)
EMT      377
BCC      .+6
JSR      PC,ERROR
```

Since the \$ form of a directive macro does not generate any executable code, it does not accept an error routine address argument.

### 3.3.4 Symbolic Offsets

Most system directive macro calls generate local symbolic offsets describing the format of the DPB. The symbols are unique to each directive, and each is assigned an index value corresponding to the offset of a given DPB element.

Because the offsets are defined symbolically, you can refer to or modify DPB elements without knowing the offset values. Symbolic offsets also eliminate the need to rewrite programs if a future release of the system changes a DPB specification.

All \$ and \$C forms of macros that generate DBPs longer than one word generate local offsets. All informational directives including the \$\$S form, generate local symbolic offsets for the parameter block returned as well.

If the program uses either the \$ or \$C form and has defined the symbol \$\$\$GLB (for example \$\$\$GLB=0), the macro generates the symbolic offsets as global symbols and does not generate the DPB itself. The purpose of this facility is to enable the use of a DPB defined in a different module. The symbol \$\$\$GLB has no effect on the expansion of \$\$S macros.

When using symbolic offsets, you should use the \$ form of directives.

### 3.3.5 Examples of Macro Calls

The following examples show the expansions of the different macro call forms:

1. The \$ form generates a DPB only, in the current p-section.

```
MRKT$ 1,5,2,MTRAP
```

generates the following code:

```
.BYTE 23.,5 ; 'MARK-TIME' DIC & DPB SIZE
.WORD 1 ; EVENT FLAG NUMBER
.WORD 5 ; TIME INTERVAL MAGNITUDE
.WORD 2 ; TIME INTERVAL UNIT (SECONDS)
.WORD MTRAP ; AST ENTRY POINT ADDRESS
```

2. The \$C form generates a DPB in p-section \$DPB\$. and generates the code to issue the directive in the specified section.

```
MRKT$C 1,5,2,MTRAP,PROG1,ERR
```

generates the following code:

```
.PSECT $DPB$.
$$$= . ; DEFINE TEMPORARY SYMBOL
.BYTE 23.,5 ; 'MARK-TIME' DIC & DPB SIZE
.WORD 1 ; EVENT FLAG NUMBER
.WORD 5 ; TIME INTERVAL MAGNITUDE
.WORD 2 ; TIME INTERVAL UNIT (SECONDS)
.WORD MTRAP ; AST ENTRY POINT ADDRESS
.PSECT PROG1 ; RETURN TO THE ORIGINAL PSECT
MOV #$$$,-(SP) ; PUSH DPB ADDRESS ON STACK
EMT 377 ; TRAP TO THE EXECUTIVE
BCC .+6 ; BRANCH ON DIRECTIVE ACCEPTANCE
JSR PC,ERR ; ELSE, CALL ERROR SERVICE ROUTINE
```

3. The \$\$S form generates code to push the DPB onto the stack and to issue the directive.

```
MRKT$$S    #1,#5,#2,R2,ERR
```

generates the following code:

```
MOV        R2,-(SP)      ; PUSH AST ENTRY POINT
MOV        #2,-(SP)      ; TIME INTERVAL UNIT (SECONDS)
MOV        #5,-(SP)      ; TIME INTERVAL MAGNITUDE
MOV        #1,-(SP)      ; EVENT FLAG NUMBER
MOV        (PC)+,-(SP)    ; AND 'MARK-TIME' DIC & DPB SIZE
.BYTE      23.,5         ; ON THE STACK
EMT        377           ; TRAP TO THE EXECUTIVE
BCC        .+6           ; BRANCH ON DIRECTIVE ACCEPTANCE
JSR        PC,ERR        ; ELSE, CALL ERROR SERVICE ROUTINE
```

4. The DIR\$ macro issues a directive that has a predefined DPB.

```
DIR$       R1,(R3)       ; DPB ALREADY DEFINED.  DPB ADDRESS IN R1.
```

generates the following code:

```
MOV        R1,-(SP)      ; PUSH DPB ADDRESS ON STACK
EMT        377           ; TRAP TO THE EXECUTIVE
BCC        .+4           ; BRANCH ON DIRECTIVE ACCEPTANCE
JSR        PC,(R3)       ; ELSE, CALL ERROR SERVICE ROUTINE
```

### 3.4 FORTRAN SUBROUTINES

The system provides a set of Fortran subroutines to perform system directive operations. In general, one subroutine is available for each directive. (Exceptions are the Mark Time and Run directives. The description of Mark Time includes both CALL MARK and CALL WAIT. The description of Run includes both CALL RUN and CALL START.)

All the subroutines described in this manual can be called by Fortran programs compiled by the FORTRAN-77 compiler.

These subroutines can also be called from programs written in the MACRO-11 assembly language by using PDP-11 FORTRAN calling sequence conventions. These conventions are described in your Fortran user's guide.

The directive descriptions in Chapter 9 describe the Fortran subroutine calls, as well as the macro calls.

#### 3.4.1 Using Subroutines

All the subroutines described in this manual are in the system object module library. You call these subroutines by including the appropriate CALL statement in the Fortran program. When the program is linked to form a task, the Task Builder first checks to see whether each specified subroutine is user defined. If a subroutine is not user defined, the Task Builder automatically searches for it in the system object module library. If the subroutine is found, it is included in the linked task.

**3.4.1.1 Optional Arguments** —Many of the subroutines described in this manual have optional arguments. In the subroutine descriptions associated with the directives, optional arguments are designated as such by being enclosed in square brackets ([ ]). An argument of this kind can be omitted if the comma that immediately follows it is retained. If the argument (or string of optional arguments) is last, it can simply be omitted, and no comma need end the argument list. For example, the format of a call to SUB could be the following:

```
CALL SUB (AA,[BB],[CC],DD[, [EE],[FF]])
```

In that event, you may omit the arguments BB, CC, EE, and FF in one of the following ways:

1. CALL SUB (AA,,,DD,,)
2. CALL SUB (AA,,,DD)

In some cases, a subroutine will use a default value for an unspecified optional argument. Such default values are noted in each subroutine description in Chapter 9.

**3.4.1.2 Task Names** —In Fortran subroutines, task names may be up to six characters long. Characters permitted in a task name are the letters A through Z, the numerals 0 through 9, and the special characters dollar sign (\$) and period (.). Task names are stored as Radix-50 code, which permits as many as three characters to be encoded in one word. (Radix-50 is described in detail in your Fortran user's guide.)

Fortran subroutine calls require that a task name be defined as a 2-word variable or array that contains the task name as Radix-50 code. This variable may be REAL, INTEGER\*4, or an INTEGER\*2 array of two elements.

This variable may be defined at program compilation time by a DATA statement, which gives the real variable an initial value (a Radix-50 constant).

For example, if a task name CCMF1 is to be used in a system directive call, the task name could be defined and used as follows:

```
DATA CCMF1/SRCCMF1/
      .
      .
      .
CALL REQUES (CCMF1)
```

A program may define task names during execution by using the IRAD50 subroutine or the RAD50 function as described in your Fortran user's guide. The equivalent data format is available for other higher-level languages.

**3.4.1.3 Integer Arguments** —All the subroutines described in this manual assume that integer arguments are INTEGER\*2-type arguments. The FORTRAN-77 system normally treats an integer variable as one storage word,

provided that its value is within the range -32768 through +32767. However, if you specify the /I4 option switch when compiling a program, ensure that all integer array arguments used in these subroutines are explicitly specified as type INTEGER\*2.

**3.4.1.4 GETADR Subroutine** —Some subroutine calls include an argument described as an integer array. The integer array contains some values that are the addresses of other variables or arrays. Since the Fortran language does not provide a means of assigning such an address as a value, you must use the GETADR subroutine as follows:

### Calling Sequence

```
CALL GETADR(ipm,[arg1],[arg2],...[argn])
```

ipm	An array of dimension n.
arg1,...argn	Arguments whose addresses are to be inserted in ipm. Arguments are inserted in the order specified. If a null argument is specified, then the corresponding entry in ipm is left unchanged. When the argument is an array name, the address of the first array element is inserted into ipm.

### Example

```
DIMENSION IBUF(80),IOSB(2),IPARAM(6)
.
.
.
CALL GETADR (IPARAM(1),IBUF(1))
IPARAM(2)=80
CALL QIO (IREAD,LUN,IEFLAG,,IOSB,IPARAM,IDSW)
.
.
.
```

In this example, CALL GETADR enables you to specify a buffer address in the CALL QIO directive (see Section 9.1.37).

### 3.4.2 The Subroutine Calls

Table 3-1 is a list of the Fortran subroutine calls (and corresponding macro calls) associated with system directives (see Chapter 9 for detailed descriptions).

For some directives, notably Mark Time (CALL MARK), both the standard Fortran subroutine call and the ISA standard call are provided. Other directives, however, are not available to Fortran tasks (for example, Specify Floating Point Exception AST [SFPA\$] and Specify SST Vector Table For Task [SVTK\$]).

Table 3-1  
 Fortran Subroutines and Corresponding Macro Calls

<i>Directive</i>	<i>Macro Call</i>	<i>Fortran Subroutine</i>
Abort Task	ABRT\$	CALL ABORT
Alter Priority	ALTP\$	CALL ALTPRI
Assign LUN	ALUN\$	CALL ASNLUN
Attach Region	ATRG\$	CALL ATRG
Cancel Time Based Initiation Requests	CSRQ\$	CALL CANALL
Cancel Mark Time Requests	CMKT\$	CALL CANMT
Clear Event Flag	CLEF\$	CALL CLREF
Create Logical Name String	CLOG\$	CALL CRELOG
Connect	CNCT\$	CALL CNCT
Create Address Window	CRAW\$	CALL CRAW
Create Region	CRRG\$	CALL CRRG
Declare Significant Event	DECL\$\$	CALL DECLAR
Disable AST Recognition	DSAR\$\$	CALL DSASTR
Disable Checkpointing	DSCP\$\$	CALL DISCKP
Detach Region	DTRG\$	CALL DTRG
Delete Logical Name String	DLOG\$	CALL DELLOG
Eliminate Address Window	ELAW\$	CALL ELAW
Emit Status	EMST\$	CALL EMST
Enable AST Recognition	ENAR\$\$	CALL ENASTR
Enable Checkpointing	ENCP\$\$	CALL ENACKP
Exit If	EXIF\$	CALL EXITIF
Exit With Status	EXST\$	CALL EXST
Extend Task	EXTK\$	CALL EXTTSK
Feature Test for Specification	FEAT\$	CALL FEAT
Get Default Directory String	GDIR\$	CALL GETDDS
Get LUN Information	GLUN\$	CALL GETLUN
Get Mapping Context	GMCX\$	CALL GMCX
Get MCR Command Line	GMCR\$	CALL GETMCR
Get Partition Parameters	GPRT\$	CALL GETPAR
Get Region Parameters	GREG\$	CALL GETREG
Get Task Parameters	GTSK\$	CALL GETTSK
Get Time Parameters	GTIM\$	CALL GETTIM
Inhibit AST Recognition	IHAR\$\$	CALL INASTR
Map Address Window	MAP\$	CALL MAP
Mark Time	MRKT\$	CALL MARK
		CALL WAIT (ISA Standard call)

Table 3-1(Cont.)

<i>Directive</i>	<i>Macro Call</i>	<i>Fortran Subroutine</i>
Queue I/O Request	QIO\$	CALL QIO
Queue I/O Request And Wait	QIOW\$	CALL WTQIO
Read All Event Flags	RDAF\$ RDXF\$	CALL READEF (Only a single, local, or common, event flag can be read by a Fortran task)
Read Single Event Flag	RDEF\$	CALL READEF
Receive By Reference	RREF\$	CALL RREF
Receive Data	RCVD\$	CALL RECEIV
Receive Data Or Exit	RCVX\$	CALL RECOEX
Receive Data Or Stop	RCST\$	CALL RCST
Request and Pass Offspring Information	RPOI\$	CALL RPOI
Request	RQST\$	CALL REQUES
Resume	RSUM\$	CALL RESUME
Run	RUN\$	CALL RUN
Send By Reference	SREF\$	CALL SREF
Send Data	SDAT\$	CALL SEND
Send, Request And Connect	SDRC\$	CALL SDRC
Send Data Request and Pass OCB	SDRP\$	CALL SDRP
Set Default Directory String	SDIR\$	CALL SETDDS
Set Event Flag	SETF\$	CALL SETEF
Set System Time	STIM\$	CALL SETTIM
Spawn	SPWN\$	CALL SPAWN
Specify Requested Exit AST	SREX\$	CALL SREX
Stop	STOP\$\$	CALL STOP
Stop For Logical OR Of Event Flags	STLO\$	CALL STLOR
Stop For Single Event Flag	STSE\$	CALL STOPFR
Suspend	SPND\$\$	CALL SUSPND
Task Exit	EXIT\$\$	CALL EXIT
Unmap Address Window	UMAP\$	CALL UNMAP
Unstop	USTP\$	CALL USTP
Variable Receive Data	VRCD\$	CALL VRCD
Variable Receive Data Or Exit	VRCX\$	CALL VRCX
Variable Receive Data Or Stop	VRCS\$	CALL VRCS
Variable Send Data	VSDA\$	CALL VSDA
Variable Send, Request and Connect	VSRC\$	CALL VSRC
Wait For Single Event Flag	WTSE\$	CALL WAITFR
Wait For Logical OR Of Event Flags	WTLO\$	CALL WFLOR
Wait For Significant Event	WSIG\$\$	CALL WFSNE
What's In My Professional	WIMP\$	CALL WIMP



Table 3-2 shows the directives that are not available as Fortran subroutines.

Table 3-2  
Directives Not Available as Subroutines

<i>Directive</i>	<i>Macro Call</i>
AST Service Exit	ASTX\$\$
Connect To Interrupt Vector	CINT\$
Specify Floating Point Exception AST	SFPA\$
Specify Receive By Reference AST	SRRAS
Specify Receive Data AST	SRDA\$
Specify SST Vector Table For Debugging Aid	SVDB\$

### 3.4.3 Error Conditions

Each subroutine call includes an optional argument that specifies the integer to receive the Directive Status Word (ids). When you specify this argument, the subroutine returns a value that indicates whether the directive operation succeeded or failed. If the directive failed, the value indicates the reason for the failure. The possible values are the same as those returned to the Directive Status Word (DSW) in MACRO-11 programs except for the two ISA calls, CALL WAIT and CALL START. The ISA calls have positive numeric error codes (see Sections 9.1.36 and 9.1.49).

### 3.4.4 AST Service Routines

The following Fortran callable routines provide support for ASTs in Fortran programs:

CALL CNCT	CALL SPAWN
CALL SDRC	CALL SREX

Whenever you specify a Fortran AST routine to one of the system library routines listed above, the AST routine is replaced by an internal routine that saves the general purpose registers and calls the specified Fortran routine using a co-routine call when the AST occurs. After the Fortran routine completes, by way of a RETURN statement, the internal routine restores the general purpose registers and issues an ASTX\$ directive.

Use great caution when coding an AST service routine in Fortran. The following types of Fortran operations may not be performed at AST state:

- *Fortran I/O of any kind*—This includes ENCODE and DECODE statements and internal file I/O. Fortran I/O is not reentrant; therefore the information in the impure data area may be destroyed.

- *Floating-point operations*—The floating-point processor's context is not saved while in AST state. Since the scientific subroutines use floating-point operations, they may not be called at AST state.
- *Traceback information in the generated code*—Use of traceback information corrupts the error recovery in the Fortran run time library. Any Fortran modules that will be called at AST state must be compiled without traceback. See your Fortran user's guide for more information.
- *Virtual array operations*—Use of virtual arrays at AST state remaps the current array such that any operations at non-AST state will not be executed correctly.
- *Subprograms*—May not be shared between AST processing and normal task processing
- *EXIT or STOP statements with files open*—Fortran flushes the task's buffers, which could be in an intermediate state. Therefore, data might be lost if any output files are open when the EXIT or STOP is executed.

You can EXIT or STOP at AST state if no output files are open.

Since the message put out by STOP uses a different mechanism from the normal Fortran I/O routines, the act of putting out this message does not corrupt impure data in the run time system. Therefore, you can issue a STOP statement at AST state unless there are output files open.

Note also the following:

- Any execution time error at AST state will corrupt the program.
- Use extreme care if the Fortran task is overlaid. Both the interface routine and the actual code of the Fortran AST routine must be located in the root segment. Any routines that are called at AST state must also be in the root segment.
- ASTs from other higher-level languages are not supported at all.

### 3.5 TASK STATES

The Executive recognizes the existence of a task only after it has been successfully installed and has an entry in the System Task Directory (STD). Once a task is known to the system, it exists in one of two states: dormant or active. Some system directives cause a task to change from one state to another.

The Executive recognizes a task immediately after it has been installed; however, the task at that point is dormant. A dormant task has an entry in the STD, but no request has been made to activate it.

A task is active from the time it is requested until the time it exits. Requesting a task means issuing the RQST\$, RUN\$, SPWN\$, SDRCS\$, VSRC\$, RPOI\$, or SDRP\$ macro. An active task is eligible for scheduling, whereas a dormant task is not.

The three substates of an active task are as follows:

1. *Ready-to-run*—A ready-to-run task competes with other tasks for CPU time on the basis of priority. The highest priority ready-to-run task obtains CPU time and thus becomes the current task.
2. *Blocked*—A blocked task is unable to compete for CPU time for synchronization reasons or because a needed resource is not available. Task priority effectively remains unchanged, allowing the task to compete for memory space.
3. *Stopped*—A stopped task is unable to compete for CPU time because of pending I/O completion, event flag(s) not set, or because the task stopped itself. When stopped, a task's priority effectively drops to zero and the task can be checkpointed by any other task, regardless of that task's priority. If an AST occurs for the stopped task, its normal task priority is restored only for the duration of the AST routine execution; once the AST is completed, task priority returns to zero.

### 3.5.1 Task State Transitions

This section describes the eight task state transitions.

*Dormant to Active*—The following directives cause the Executive to activate a dormant task:

- RUN\$
- RQST\$
- SPWN\$
- SDRCS\$
- VSRC\$
- RPOI\$
- SDRP\$

*Ready-to-Run to Blocked*—The following events cause an active, ready-to-run task to become blocked:

- A SPND\$ directive
- An unsatisfied Wait For condition
- Checkpointing of a task out of memory by the Executive

*Ready-to-Run to Stopped*—The following events cause an active, ready-to-run task to become stopped:

- A STOP\$\$ directive is executed, or an RCST\$, SDRP\$, or VRCS\$ directive is issued when no data packet is available.
- An unsatisfied Stop For condition.
- An unsatisfied Wait For condition while the task has outstanding buffered I/O.

*Blocked to Ready-to-Run*—The following events return a blocked task to the ready-to-run state:

- An RSUM\$ directive issued by another task
- A Wait For condition is satisfied
- The Executive reads a checkpointed task into memory

*Stopped to Ready-to-Run*—The following events return a stopped task to the ready-to-run state, depending upon how the task became stopped:

- A task stopped by the STOP\$, RCST\$, or VRCS\$ directive becomes unstopped by USTP\$ directive execution.
- A Wait For condition is satisfied for a task with outstanding buffered I/O.
- A task stopped for an event flag becomes unstopped when the specified event flag becomes set.

*Active to Dormant*—The following events cause an active task to become dormant:

- An EXIT\$\$, EXIF\$, RCVX\$, or VRCX\$ directive, or a RREF\$ directive that specifies the exit option
- An ABRT\$ directive
- A Synchronous System Trap (SST) for which a task has not specified a service routine

*Blocked to Stopped*—The following event causes a task that is blocked due to an unsatisfied Wait For condition to become stopped:

- The task initiates buffered I/O at AST state and then exits from AST state.

*Stopped to Blocked*—The following event causes a task that is stopped due to an unsatisfied Wait For condition and outstanding buffered I/O to return to a blocked state:

- Completion of all outstanding buffered I/O

### 3.6 DIRECTIVE RESTRICTIONS FOR NONPRIVILEGED TASKS

Nonprivileged tasks cannot issue certain Executive directives, except for those shown in Table 3-3:

Table 3-3  
System Directives That Can Be Issued by Nonprivileged Tasks

<i>Directive</i>	<i>Macro Call</i>	<i>Comments</i>
Abort Task	ABRT\$	A nonprivileged task can only abort tasks with the same TI: as the task issuing the directive.
Alter Priority	ALTP\$	A nonprivileged task can only alter its own priority to values less than or equal to the task's installed priority.
Cancel Time Based Initiation Requests	CSRQ\$	Cannot be issued by a nonprivileged task except for tasks with the same TI: as the issuing task.
Switch State	SWST\$	Cannot be issued by a nonprivileged task.

### 3.7 DIRECTIVE CATEGORIES

This section groups the directives by function into the following eight categories:

1. Task execution control
2. Task status control
3. Informational
4. Event-associated
5. Trap-associated
6. I/O- and intertask communications-related
7. Memory management
8. Parent/offspring tasking

#### 3.7.1 Task Execution Control Directives

The task execution control directives deal principally with starting and stopping tasks. Each of these directives (except Extend Task) results in a change of the task's state (unless the task is already in the state being requested). Table 3-4 shows the task execution control directives.

Table 3-4  
Task Execution Control Directives

<i>Macro</i>	<i>Directive Name</i>
ABRT\$	Abort Task
CSRQ\$	Cancel Time Based Initiation Requests
EXIT\$\$	Task Exit (\$\$ form recommended)
EXTK\$	Extend Task
RQST\$	Request Task
SPND\$\$	Suspend (\$\$ form recommended)
SWST\$	Switch State

### 3.7.2 Task Status Control Directives

Two task status control directives alter the checkpointable attribute of a task. A third directive changes the running priority of an active task. Table 3-5 shows the task status control directives.

### 3.7.3 Informational Directives

Several directives provide the issuing task with system information and parameters such as the time of day, the task parameters, the console switch settings, and partition or region parameters. Table 3-6 shows the informational directives.

### 3.7.4 Event-Associated Directives

The event and event flag directives provide inter- and intratask synchronization and signaling and the means to set the system time. You must use these directives carefully since software faults resulting from erroneous signaling and synchronization are often obscure and difficult to isolate. Table 3-7 shows the event-associated directives.

### 3.7.5 Trap-Associated Directives

The trap-associated directives provide trap facilities that allow transfer of control (software interrupts) to the executing tasks. Table 3-8 shows the trap-associated directives.

Table 3-5  
Task Status Control Directives

<i>Macro</i>	<i>Directive Name</i>
ALTP\$	Alter Priority
DSCP\$\$	Disable Checkpointing (\$\$ form recommended)
ENCP\$\$	Enable Checkpointing (\$\$ form recommended)

Table 3-6  
Informational Directives

<i>Macro</i>	<i>Directive Name</i>
GPRT\$	Get Partition Parameters
GREG\$	Get Region Parameters
GTIM\$	Get Time Parameters
GTSK\$	Get Task Parameters

Table 3-7  
Event-Associated Directives

<i>Macro</i>	<i>Directive Name</i>
CLEF\$	Clear Event Flag
CMKT\$	Cancel Mark Time Requests
DECL\$\$	Declare Significant Event (\$\$ form recommended)
EXIF\$	Exit If
MRKT\$	Mark Time
RDEF\$	Read Single Event Flag
SETF\$	Set Event Flag
STIM\$	Set System Time
STLO\$	Stop For Logical 'OR' of Event Flags
STOP\$\$	Stop (\$\$ form recommended)
STSE\$	Stop For Single Event Flag
USTP\$	Unstop
WSIG\$\$	Wait For Significant Event (\$\$ form recommended)
WTLO\$	Wait For Logical OR Of Event Flags
WTSE\$	Wait For Single Event Flag

Table 3-8  
Trap-Associated Directives

<i>Macro</i>	<i>Directive Name</i>
ASTX\$\$	AST Service Exit (\$S form recommended)
DSAR\$\$	Disable AST Recognition (\$S form recommended)
ENAR\$\$	Enable AST Recognition (\$S form recommended)
IHAR\$\$	Inhibit AST Recognition (\$S form recommended)
SFPA\$	Specify Floating Point Processor Exception AST
SRDA\$	Specify Receive Data AST
SREA\$	Specify Requested Exit AST
SREX\$	Specify Requested Exit AST (extended)
SRRA\$	Specify Receive-By-Reference AST
SVDB\$	Specify SST Vector Table For Debugging Aid
SVTK\$	Specify SST Vector Table For Task

### 3.7.6 I/O- and Intertask Communications-Related Directives

The I/O- and intertask communications-related directives allow tasks to access I/O devices at the driver interface level or interrupt level, to communicate with other tasks in the system, and to retrieve the MCR command line used to start the task. Table 3-9 shows the I/O- and intertask communications-related directives.

### 3.7.7 Memory Management Directives

The memory management directives allow a task to manipulate its virtual and logical address space, and to set up and control dynamically the window-to-region mapping assignments. The directives also provide the means by which tasks can share and pass references to data and routines. Table 3-10 shows the memory management directives.

### 3.7.8 Parent/Offspring Tasking Directives

Parent/offspring tasking directives permit tasks to start other tasks, and to connect to other tasks in order to receive status information. Table 3-11 shows the parent/offspring tasking directives.



Table 3-9  
I/O- and Intertask Communications-Related Directives

<i>Macro</i>	<i>Directive Name</i>
ALUN\$	Assign LUN
CINT\$	Connect To Interrupt Vector
CLOG\$	Create Logical Name String
DLOG\$	Delete Logical Name String
GDIR\$	Get Default Directory String
GLUN\$	Get LUN Information
GMCR\$	Get MCR Command Line
QIO\$	Queue I/O Request
QIOW\$	Queue I/O Request And Wait
RCVD\$	Receive Data
RCVX\$	Receive Data Or Exit
SDAT\$	Send Data
SDIR\$	Set Default Directory String
VRCD\$	Variable Receive Data
VRC\$	Variable Receive Data Or Stop
VRCX\$	Variable Receive Data Or Exit
VSDA\$	Variable Send Data

Table 3-10  
Memory Management Directives

<i>Macro</i>	<i>Directive Name</i>
ATRG\$	Attach Region
CRAW\$	Create Address Window
CRRG\$	Create Region
DTRG\$	Detach Region
ELAW\$	Eliminate Address Window
GMCX\$	Get Mapping Context
MAP\$	Map Address Window
RREF\$	Receive By Reference
SREF\$	Send By Reference
UMAP\$	Unmap Address Window

Table 3-11  
Parent/Offspring Tasking Directives

<i>Macro</i>	<i>Directive Name</i>
CNCT\$	Connect
EMST\$	Emit Status
EXST\$	Exit With Status
RPOI\$	Request and Pass Offspring Information
SDRC\$	Send, Request, And Connect
SDRP\$	Send Data, Request and Pass OCB
SPWN\$	Spawn
VSRC\$	Variable Send, Request, and Connect

### 3.8 DIRECTIVE CONVENTIONS

The following are conventions for using system directives:

1. In MACRO-11 programs, unless a number is followed by a decimal point (.), the system assumes the number to be octal.  
In Fortran programs, use INTEGER\*2 type unless the directive description states otherwise.
2. In MACRO-11 programs, task and partition names can be from one through six characters long and should be represented as two words in Radix-50 form.  
In Fortran programs, specify task and partition names by a variable of type REAL (single precision) that contains the task or partition name in Radix-50 form. To establish Radix-50 representation, either use the DATA statement at compile time, or use the IRAD50 subprogram or RAD50 function at run time.
3. Device names are two characters long and are represented by one word of ASCII code.
4. Some directive descriptions state that a certain parameter must be provided even though the system ignores it. Such parameters are included for future extension to the system.
5. In the directive descriptions, square brackets ([ ]) enclose optional parameters or arguments. To omit optional items, either use an empty (null) field in the parameter list or omit a trailing optional parameter.
6. Logical Unit Numbers (LUNs) can range from 1 through 255<sub>10</sub>.
7. Event flag numbers range from 1 through 64<sub>10</sub>. Numbers from 1 through 32<sub>10</sub> denote local flags. Numbers from 33 through 64 denote common flags.

Note that the Executive preserves all task registers when a task issues a directive.

## CHAPTER 4

# LOGICAL NAMES

---

A logical name is a character string used to represent a file or device by other than its specific physical name. Logical names allow you to write programs that are independent of the physical devices or files used in input or output operations. The CLOG\$ and DLOG\$ directives perform the following logical name functions:

- Create a logical name string (CLOG\$)
- Delete a logical name string (DLOG\$)

(See the description of these directives later in this chapter.)

### 4.1 LOGICAL NAMES AND EQUIVALENCE NAMES

A logical name string always refers to an associated equivalence name string. The system provides a logical name facility that translates a logical name and returns its equivalence string. Within the strict context of the logical name facility, the logical name and its equivalence name are simply byte strings. The only restrictions to logical name strings and equivalence names strings are:

- The string length must not exceed  $255_{10}$  bytes.
- There must be an equivalence name string for each logical name string entered in the logical name table.

#### 4.1.1 The Logical Name Table

The system stores logical name strings and their equivalence strings in a single logical name table. The logical name table contains names that cooperating tasks can use. The system uses this table when translating logical names.

### 4.1.2 Duplicate Logical Names

The logical name table can contain multiple equivalence strings for the same logical name. However, each duplicate logical name must be distinguished by a unique modifier. The mod argument to the CLOG\$ directive serves that purpose, allowing a maximum of  $255_{10}$  duplicate logical names. See the description of the CLOG\$ directive in Chapter 9. You may specify mod argument identifier values 128 through  $256_{10}$ ; identifier values 0 through 127 are reserved for system use.

Note: Duplicate logical names are possible only when the user tasks handle the logical name translations. Within the context of the system software (such as RMS and volume mounting procedures), the only recognized value for the mod argument is 0.

If a newly created logical name duplicates an existing logical name having the same value for the mod argument, the system supersedes the old logical name definition with the new one.

Any number of logical names can have the same equivalence name. Furthermore, the number of logical names possible is limited only by the amount of available secondary pool in the system.

## 4.2 RMS TRANSLATION OF LOGICAL NAMES

As part of I/O processing in program execution, RMS translates logical names and returns their equivalence names. The following conventions govern RMS translation of logical names:

- RMS translates only those logical names occurring within the context of a valid device specification.
- RMS continues to do translations of logical name strings until it encounters an equivalence name string beginning with an underscore (`_`), until it fails to translate a string, until it encounters an equivalence name string not ending with a colon (`:`), or until it reaches the maximum number of translations allowed.
- RMS does a maximum of eight translations for a given logical name. If the number of logical name translations exceeds the maximum, RMS issues an error.

### 4.2.1 RMS and Default Directories

The system provides a special case of logical names known as a default directory. The default directory is a character string stored in the system secondary pool. If RMS encounters an input string with no specified directory, or if the input string contains a pair of closed empty brackets—the explicit request for the default directory—RMS returns the default directory string.

### 4.3 FILES-11 ACP USE OF LOGICAL NAMES

The Files-11 ACP creates logical names when it mounts a file-structured disk. The ACP creates a logical name using the volume label specified at the time the volume was initialized. It creates an equivalence string that returns the name of the physical device on which the volume is mounted.

The ACP also creates a second logical name when mounting a file-structured disk using the physical device name as the logical name and the volume label as the equivalence name. For example:

```
LOGICAL NAME      FINANCE:  DDnnn:
EQUIVALENCE NAME +DDnnn:  FINANCE:
```

An application program can reference the disk with the volume label FINANCE by using the logical name FINANCE:. RMS translates the logical name to determine the actual physical device. Similarly, an application programmer can use the logical name scheme in the example to determine the volume that is currently mounted.

### 4.4 LOGICAL NAME CREATION

Use the CLOG\$ directive to create a logical name string and the associated equivalence name string (see Chapter 9). The length of each logical name string can be a maximum of 255<sub>10</sub> characters (bytes). Creation of the logical name string requires the use of the secondary pool which is of limited size.

The following example shows how to create a logical name with the CLOG\$ directive.

```
      .MCALL      CLOG$,DIR$
LNAME:  .ASCII   /EXPENSES:/ ;LOGICAL NAME STRING
LNAMSZ=  .-LNAME                ;SIZE OF LOGICAL NAME STRING
ENAME:  .ASCII   /FINANCE:/ ;EQUIVALENCE NAME STRING
ENAMSZ=  .-ENAM                ;DEFINE SIZE OF EQUIVALENCE
                                   ;NAME STRING
      .EVEN
NAMVOL:  CLOG$      ,LT.USR,LNAME,LNAMSZ,ENAME,ENAMSZ
START:   DIR$      #NAMVOL      ;CREATE LOGICAL NAME
```

### 4.5 LOGICAL NAME TRANSLATION

A subroutine called PROLOG is available in POSSUM that allows the translation of logical names. This subroutine is callable from MACRO and high-level languages. (See Chapter 8.)

### 4.6 LOGICAL NAME DELETION

Use the DLOG\$ directive to delete entries from the logical name table. When you code a call to the DLOG\$ directive, you can delete a single logical name from the table, or you can delete all the logical names in the table.

The following example deletes a single logical name entry from the logical name table:

```

      .MCALL      DLOG$,DIR$
NAME:  .ASCII    /TMONK/
NAMESZ= .-NAME
      .EVEN
NAMDEL: DLOG$    ,LT.USR,NAME, NAMESZ
START:  DIR$     #NAMDEL      ;DELETE LOGICAL NAME
      .
      .
      .

```

On the other hand, the example below deletes all the logical name entries in the user logical name table:

```

      .MCALL      DLOG$,DIR$
DELALL: DLOG$    ,LT.USR
START:  DIR$     #DELALL      ;DELETE LOGICAL NAME
      .
      .
      .

```

See the DLOG\$ directive description in Section 9.1.14 for more details on deleting logical names.

#### 4.7 SETTING UP A DEFAULT DIRECTORY STRING

Use the SDIR\$ macro to establish a default directory.

The following example shows how to use the SDIR\$ macro to set up a default directory string:

```

      .MCALL      SDIR$,DIR$
DDSNAM: .ASCII    /[SOLOS]/
DDSSZ=  .-DDSNAM
      .EVEN
SETNAM:  SDIR$    ,DDSNAM,DDSSZ
START:  DIR$     #SETNAM      ;ESTABLISH DEFAULT DIRECTORY
      .
      .
      .

```

#### 4.8 RETRIEVING A DEFAULT DIRECTORY STRING

Use the GDIR\$ directive to retrieve a default directory string. The system returns the default directory string to the specified user buffer along with the length of the string.

The following example shows how to use the GDIR\$ macro to retrieve the default directory string:

```

      .MCALL      GDIR$,DIR$
DDSNAM:  .BLKB      100.          ;DEFINE BUFFER FOR DEFAULT
                                           ;DIRECTORY STRING
DDSSZ=   .-DDSNAM          ;CALCULATE BUFFER SIZE
      .EVEN
GETNAM:  GDIR$      ,DDSNAM,DDSSZ
START:   DIR$      #GETNAM      ;GET DEFAULT DIRECTORY STRING
      .
      .
      .

```





# CHAPTER 5

## SIGNIFICANT EVENTS, EVENT FLAGS, SYSTEM TRAPS, AND STOP-BIT SYNCHRONIZATION

---

This chapter introduces the concept of significant events and describes the ways in which your code can make use of event flags, synchronous and asynchronous system traps, and stop-bit synchronization.

### 5.1 SIGNIFICANT EVENTS

A significant event is a change in system status that causes the Executive to reevaluate the eligibility of all active tasks to run. (For some significant events, specifically those in which the current task becomes ineligible to run, only those tasks of lower priority are examined.) A significant event is usually caused (either directly or indirectly) by a system directive issued from within a task. (All of the system directives named in this chapter are described in detail in Chapter 9.)

Significant events include the following:

- An I/O completion
- A task exit
- The execution of a Send Data directive
- The execution of a Send Data, Request and Pass OCB directive
- The execution of a Send, Request, and Connect directive
- The execution of a Send By Reference or a Receive By Reference directive
- The execution of an Alter Priority directive
- The removal of an entry from the clock queue (for example, resulting from the execution of a Mark Time directive or the issuance of a rescheduling request)
- The execution of a Declare Significant Event directive

- ❑ The execution of the round-robin scheduling algorithm at the end of a round-robin scheduling interval
- ❑ The execution of an Exit, an Exit With Status, or an Emit Status directive

## 5.2 EVENT FLAGS

Event flags are a means by which tasks recognize specific events. (Tasks also use Asynchronous System Traps (ASTs) to recognize specific events. See Section 5.3.3.) In requesting a system operation (such as an I/O transfer), a task may associate an event flag with the completion of the operation. When the event occurs, the Executive sets the specified flag. Several examples later in this section describe how tasks can use event flags to coordinate task execution.

Sixty-four event flags are available to enable tasks to distinguish one event from another. Each event flag has a corresponding unique Event Flag Number (EFN). Numbers 1 through 32 form a group of local flags that are unique to each task and are set or cleared as a result of that task's operation. Numbers 33 through 64 form a second group of flags that are common to all tasks, hence their name "common flags." Common flags may be set or cleared as a result of any task's operation. The last eight flags in each group, local flags (25 through 32) and common flags (57 through 64), are reserved for use by the system.

Tasks can use the common flags for intertask communication or their own local event flags internally. They can set, clear, and test event flags by using Set Event Flag (SETF\$), Clear Event Flag (CLEF\$), and Read All Event Flags (RDAF\$) directives.

**Caution:** Take great care when setting or clearing event flags, especially common flags. Erroneous or multiple setting and clearing of event flags can result in obscure software faults. A typical application program can be written without explicitly accessing or modifying event flags, since many of the directives can implicitly perform these functions. The Send Data (SDAT\$), Mark Time (MRKT\$), and the I/O operations directives can all implicitly alter an event flag.

Examples 1 and 2 illustrate the use of common event flags (33 through 64) to synchronize task execution. Examples 3 and 4 illustrate the use of local flags (1 through 32).

### Example 1

Task B clears common event flag 35 and then blocks itself by issuing a Wait For directive that specifies common event flag 35.

Subsequently another task, Task A, specifies event flag 35 in a Set Event Flag directive to inform Task B that it may proceed. Task A then issues a Declare Significant Event directive to ensure that the Executive will schedule Task B.

**Example 2**

To synchronize the transmission of data between Tasks A and B, Task A specifies Task B and common event flag 42 in a Send Data directive.

Task B has specified flag 42 in a Wait For directive. When Task A's Send Data directive has caused the Executive to set flag 42 and to cause a significant event, Task B proceeds and issues a Receive Data directive because its Wait For condition has been satisfied.

**Example 3**

A task contains a Queue I/O Request directive and an associated Wait For directive; both directives specify the same local event flag. When the task queues its I/O request, the Executive clears the local flag. If the requested I/O is incomplete when the task issues the Wait For directive, the Executive blocks the task.

When the requested I/O is completed, the Executive sets the local flag and causes a significant event. The task then resumes its execution at the instruction that follows the Wait For directive. Using the local event flag in this manner ensures that the task does not manipulate incoming data until the transfer is complete.

**Example 4**

A task specifies the same local event flag in a Mark Time and an associated Wait For directive. When the Mark Time directive is issued, the Executive first clears the local flag and subsequently sets it when the indicated time interval has elapsed.

If the task issues the Wait For directive before the local flag is set, the Executive blocks the task, which resumes when the flag is set at the end of the proper time interval. If the flag has been set first, the directive is a no-op and the task is not blocked.

Specifying an event flag does not mean that a Wait For directive must be issued. Event flag testing can be performed at any time. The purpose of a Wait For directive is to stop task execution until an indicated event occurs. Hence, it is not necessary to issue a Wait For directive immediately following a Queue I/O Request directive or a Mark Time directive.

If a task issues a Wait For directive that specifies an event flag that is already set, the blocking condition is immediately satisfied and the Executive immediately returns control to the task.

Tasks can issue Stop For directives instead of Wait For directives. When this is done, an event flag condition not satisfied will result in the task's being stopped instead of being blocked until the event flag(s) is set. A task that is blocked still competes for memory resources at its running priority. A task that is stopped competes for memory resources at priority 0.

The simplest way to test a single event flag is to issue the directive CLEF\$ or SETF\$. Both these directives can cause the following return codes:

IS.CLR	Flag was previously clear
IS.SET	Flag was previously set

For example, if a set common event flag indicates the completion of an operation, a task can issue the CLEF\$ directive both to read the event flag and simultaneously to reset it for the next operation. If the event flag was previously clear (the current operation was incomplete), the flag remains clear.

### 5.3 SYSTEM TRAPS

System traps are transfers of control (also called software interrupts) that provide tasks with a means of monitoring and reacting to events. The Executive initiates system traps when certain events occur. The trap transfers control to the task associated with the event and gives the task the opportunity to service the event by entering a user-written routine.

There are two kinds of system traps:

1. *Synchronous System Traps (SSTs)*—SSTs detect events directly associated with the execution of program instructions. They are synchronous because they always recur at the same point in the program when trap-causing instructions occur. For example, an illegal instruction causes an SST.
2. *Asynchronous System Traps (ASTs)*—ASTs detect events that occur asynchronously to the task's execution. That is, the task has no direct control over the precise time that the event—and therefore the trap—may occur. For example, the completion of an I/O transfer may cause an AST to occur.

A task that uses the system trap facility issues system directives that establish entry points for user-written service routines. Entry points for SSTs are specified in a single table. AST entry points are set by individual directives for each kind of AST. When a trap condition occurs, the task automatically enters the appropriate routine (if its entry point has been specified).

#### 5.3.1 Synchronous System Traps (SSTs)

SSTs can detect the execution of:

- Illegal instructions
- Instructions with invalid addresses
- Trap instructions (TRAP, EMT, IOT, BPT)

The user can set up an SST vector table, containing one entry per SST type. Each entry is the address of an SST routine that services a particular type of SST (a routine that services illegal instructions, for example). When an SST occurs, the Executive transfers control to the routine for that type of SST. If a corresponding routine is not specified in the table, the task is aborted.

The SST routine enables the user to process the failure and then return to the interrupted code. Note that if a debugging aid and the user's task both have an SST vector enabled for a given condition, the debugging aid vector is referenced first to determine the service routine address.

SST routines must always be reentrant if there is a possibility that an SST can occur within the SST routine itself. Although the Executive initiates SSTs, the execution of the related service routines is indistinguishable from the task's normal execution. An AST or another SST can therefore interrupt an SST routine.

### 5.3.2 SST Service Routines

The Executive initiates SST service routines by pushing the task's Processor Status (PS), Program Counter (PC), and trap-specific parameters onto the task's stack. After removing the trap-specific parameters, the service routine returns control to the task by issuing an RTI or RTT instruction. Note that the task's general purpose registers R0 through R5 and SP are not saved. If the SST routine makes use of them, it must save and restore them itself.

To the Executive, SST routine execution is indistinguishable from normal task execution, so that all directive services are available to an SST routine. An SST routine can remove the interrupted PS and PC from the stack and transfer control anywhere in the task; the routine does not have to return control to the point of interruption. Note that any operations performed by the routine (such as the modification of registers or the DSW, or the setting or clearing of event flags) remain in effect when the routine eventually returns control to the task.

A trap vector table within the task contains all the service routine entry points. You can specify the SST vector table by means of the Specify SST Vector Table For Task directive or the Specify SST Vector For Debugging Aid directive. The trap vector table has the following format shown in Table 5-1.

Table 5-1  
Trap Vector Table

<i>Word</i>	<i>Offset</i>	<i>Associated Vector</i>	<i>Trap</i>
0	S.COAD	4	Nonexistent memory error
1	S.CSGF	250	Memory protect violation
2	S.CBPT	14	T-bit trap or execution of a BPT instruction
3	S.CIOT	20	Execution of an IOT instruction
4	S.CILI	10	Execution of a reserved instruction
5	S.CEMT	30	Execution of a non-RSX EMT instruction
6	S.CTRP	34	Execution of a TRAP instruction

Depending on the reason for the SST, the task's stack may also contain additional information, as follows:

Memory protect violation (complete stack)

SP+10	PS
SP+06	PC
SP+04	Memory protect status register (SR0)
SP+02	Virtual PC of the faulting instruction (SR2)
SP+00	Instruction backup register (SR1) <sup>1</sup>

TRAP instruction or EMT other than 377 (and 376 in the case of unmapped tasks and mapped privileged tasks) (complete stack)

SP+04	PS
SP+02	PC
SP+00	Instruction operand (low-order byte) multiplied by 2, non-sign-extended

All items except the PS and PC must be removed from the stack before the SST service routine exits.

### 5.3.3 Asynchronous System Traps (ASTs)

The primary purpose of an AST is to inform the task that a certain event has occurred (for example, the completion of an I/O operation). As soon as the task has serviced the event, it can return to the interrupted code.

Some directives can specify both an event flag and an AST; with these directives, ASTs can be used as an alternative to event flags or the two can be used together. Therefore, you can specify the same AST routine for several directives, each with a different event flag. Thus, when the Executive passes control to the AST routine, the event flag can determine the action required.

AST service routines must save and restore all registers used. If the registers are not restored after an AST has occurred, the task's subsequent execution may be unpredictable.

Although it cannot distinguish between execution of an SST routine and task execution, the Executive is aware that a task is executing an AST routine. An AST routine can be interrupted by an SST routine, but not by another AST routine.

---

1. For details of SR0, SR1, and SR2, see the section on the memory management unit in the microcomputers processor handbook.

The following notes describe general characteristics and uses of ASTs:

- If an AST occurs while the related task is executing, the task is interrupted so that the AST service routine can be executed.
- If an AST occurs while another AST is being processed, the Executive queues the latest AST (First-In-First-Out or FIFO). The task then processes the next AST in the queue when the current AST service is complete (unless AST recognition was disabled by the AST service routine).
- If an AST suspends a task, the task remains stopped or suspended after the AST routine is executed, unless the task is explicitly resumed or unstopped either by the AST service routine itself, or by another task.
- If an AST occurs while the related task is waiting (or stopped) for an event flag to be set (a Wait For or Stop For directive), the task continues to wait after execution of the AST service routine unless the event flag is set upon AST exit.
- If an AST occurs for a checkpointed task, the Executive queues the AST (FIFO), brings the task into memory, and then activates the AST.
- The Executive allocates the necessary dynamic memory when an AST is specified. Thus, no AST condition lacks dynamic memory for data storage when it actually occurs. The AST reuses the storage allocated for I/O and Mark Time directives. Therefore, no additional dynamic storage is required.
- Two directives, Disable AST Recognition and Enable AST Recognition, allow a program to queue ASTs for subsequent execution during critical sections of code. (A critical section might be one that accesses data bases also accessed by AST service routines, for example.) If ASTs occur while AST recognition is disabled, they are queued (FIFO) and then processed when AST recognition is enabled.

#### 5.3.4 AST Service Routines

When an AST occurs, the Executive pushes the task's Wait For mask word, the DSW, the PS, and the PC onto the task's stack. This information saves the state of the task so that the AST service routine has access to all the available Executive services. The preserved Wait For mask word allows the AST routines to establish the conditions necessary to unblock the waiting task. Depending on the reason for the AST, the stack may also contain additional parameters. Note that the task's general purpose registers R0 through R5 and SP are not saved. If the AST service routine makes use of them, it must save and restore them itself.

The Wait For mask word comes from the offset T.EFLM in the task's Task Control Block (TCB). The value of the Wait For mask word and the event flag range to which it corresponds depend on the last Wait For or Stop For directive issued by the task. For example, if the last such directive issued was Wait For Single Event Flag 42, the mask word has a value of 1000<sub>8</sub> and the event flag range is from 33 through 48. Bit 0 of the mask word represents flag 33, bit 1 represents flag 34, and so on.

The Wait For mask word is meaningless if the task has not issued any type of Wait For or Stop For directive.

Your code should not attempt to modify the Wait For mask while in the AST routine. For example, putting a zero in the Wait For mask results in an uncleared Wait For state.

After processing an AST, the task must remove the trap-dependent parameters from its stack; that is, everything from the top of the stack down to, but not including, the task's Directive Status Word. It must then issue an AST Service Exit directive with the stack set as indicated in the description of that directive (see Section 9.1.4). When the AST service routine exits, it returns control to one of two places: another AST or the original task.

There are 8 variations on the format of the task's stack, as follows:

1. If a task needs to be notified when a Floating Point Processor exception trap occurs, it issues a Specify Floating Point Processor Exception AST directive. If the task specifies this directive, an AST occurs when a Floating Point Processor exception trap occurs. The stack contains the following values:

SP+12	Event flag mask word
SP+10	PS of task prior to AST
SP+06	PC of task prior to AST
SP+04	Task's DSW
SP+02	Floating exception code
SP+00	Floating exception address

Note: Refer to the *Microcomputers and Memories* handbook for a description of the FPU exception code values.

2. If a task needs to be notified when it receives either a message or a reference to a common area, it issues either a Specify Receive Data AST or a Specify Receive By Reference AST directive. An AST occurs when the message or common reference is sent to the task. The stack contains the following values:

SP+06	Event flag mask word
SP+04	PS of task prior to AST
SP+02	PC of task prior to AST
SP+00	Task's DSW

3. When a task queues an I/O request and specifies an appropriate AST service entry point, an AST occurs upon completion of the I/O request. The task's stack contains the following values:

SP+10	Event flag mask word
SP+06	PS of task prior to AST
SP+04	PC of task prior to AST



- |  |       |   |
|--|-------|---|
|  | SP+02 | Task's DSW  |
|  | SP+00 | Address of I/O status block for I/O request (or zero if none was specified) |
4. When a task issues a Mark Time directive and specifies an appropriate AST service entry point, an AST occurs when the indicated time interval has elapsed. The task's stack contains the following values:
- |  |       |   |
|--|-------|---|
|  | SP+10 | Event flag mask word                              |
|  | SP+06 | PS of task prior to AST                           |
|  | SP+04 | PC of task prior to AST                           |
|  | SP+02 | Task's DSW  |
|  | SP+00 | Event flag number (or zero if none was specified) |
5. An offspring task, connected by a Spawn, Connect, or Send, Request And Connect directive, returns status to the connected (parent) task(s) upon exiting by the Exit AST. The parent task's stack contains the following values:
- |  |       |                              |
|--|-------|------------------------------|
|  | SP+10 | Event flag mask word         |
|  | SP+06 | PS of task prior to AST      |
|  | SP+04 | PC of task prior to AST      |
|  | SP+02 | Task's DSW                   |
|  | SP+00 | Address of exit status block |
6. If a directive aborts a task when the Specify Requested Exit AST (SREA\$) is in effect, the abort AST is entered. The task's stack contains the following values:
- |  |       |                         |
|--|-------|-------------------------|
|  | SP+06 | Event flag mask word    |
|  | SP+04 | PS of task prior to AST |
|  | SP+02 | PC of task prior to AST |
|  | SP+00 | Task's DSW              |
7. If a directive aborts a task when the Extended Specify Requested Exit AST (SREX\$) is in effect, the abort AST is entered. The task's stack contains the following values:
- |  |       |   |
|--|-------|---|
|  | SP+12 | Event flag mask word                        |
|  | SP+10 | PS of task prior to AST                     |
|  | SP+06 | PC of task prior to AST                     |
|  | SP+04 | DSW of task prior to AST                    |
|  | SP+02 | Trap dependent parameter                    |
|  | SP+00 | Number of bytes to add to SP to clean stack |

8. If a task issues a QIO IO.ATA function to the terminal driver, unsolicited terminal input will cause entry into the AST service routine. Upon entry into the routine, the task's stack contains the following values:

SP+10	Event flag mask word
SP+06	PS of task prior to AST
SP+04	PC of task prior to AST
SP+02	Task's DSW
SP+00	Unsolicited character in low byte; parameter 2 in the high byte

#### 5.4 STOP-BIT SYNCHRONIZATION

Stop-bit synchronization allows tasks to be checkpointed during terminal (buffered) I/O or while waiting for an event to occur (for example, an event flag to be set or an Unstop directive to be issued). You can control synchronization between tasks by the setting of the task's Task Control Block (TCB) stop bit.

When the task's stop bit is set, the task is blocked from further execution, its priority for memory allocation effectively drops to zero, and it may be checkpointed by any other task in the system, regardless of priority. If checkpointed, the task remains out of memory until its stop bit is cleared, at which time the task becomes unstopped, its normal priority for memory allocation becomes restored, and it is considered for memory allocation based on the restored priority.

If the stopped task receives an AST, the task becomes unstopped until it exits the AST routine. Memory allocation for the task during the AST routine is based on the task's priority before the stopped state. Note that a task cannot be stopped when an AST is in progress, but the AST routine can issue either an Unstop or Set Event Flag directive to reference the task. This causes the task to remain unstopped after it issues the AST Service Exit directive.

There are three ways in which a nonprivileged task can be stopped and three corresponding ways it can become unstopped. Only one method for stopping a task can be applied at a time.

1. A task is stopped whenever it is in a Wait For state and has outstanding buffered I/O. A task is unstopped when the buffered I/O is completed or when the Wait For condition is satisfied.
2. You can stop a task for event flag by issuing the Stop For Single Event Flag directive or the Stop For Logical OR Of Event Flags directive. In this case, the task can only be unstopped by setting the specified event flag.
3. You can stop a task by issuing the Stop or the Receive Data Or Stop directive. In this case, the task can only be unstopped by issuing the Unstop directive.

You cannot stop a task when an AST is in progress (AST state). Any directives that can cause a task to become stopped are illegal at the AST state.

When a task is stopped for any reason at the task state, the task can still receive ASTs. If the task is checkpointed, it becomes eligible for entrance back into memory when an AST is queued for it. The task retains its normal priority in memory while it is at the AST state or has ASTs queued. Once the task has exited the AST routine with no other ASTs queued, the task is again stopped and effectively has zero priority for memory allocation.

You can use five directives for stop-bit synchronization:

1. *Stop*—This directive stops the issuing task and cannot be issued at the AST state.
2. *Receive Data Or Stop and Variable Receive Data Or Stop*—These directives attempt to dequeue send data packets from the specified task (or any task if none is specified). If there is no such packet to be dequeued, the issuing task is stopped. These directives cannot be issued at the AST state.
3. *Stop For Logical OR Of Event Flags* —This directive stops the issuing task until the specified flags in the specified group of local event flags become set. If any of the specified event flags are already set, the task does not become stopped. This directive cannot be issued at the AST state.
4. *Stop For Single Event Flag*—This directive stops the issuing task until the indicated local event flag becomes set. If the specified event flag is already set, the task does not become stopped. This directive cannot be issued at the AST state.
5. *Unstop*—This directive unstops a task that has become stopped by the Stop or Receive Data Or Stop directive.



## CHAPTER 6

# PARENT/OFFSPRING TASKING

---

Parent/offspring tasking allows you to establish and control the relationships between a governing (parent) task and any subordinate (offspring) tasks. A parent task starts or connects to another an offspring task.

One application for the parent-offspring task relationship is a multitask application. In such an application, the main task controlling the application requires other tasks to perform subfunctions for the application. With parent/offspring tasking, you can set up the necessary relationships between the parent task and its offspring to control processing.

Starting (or activating) offspring tasks is called “spawning” Spawning also includes the ability to establish task communications; a parent task can be notified when an offspring task exits and can receive status information from the offspring task.

Status returned from an offspring task to a parent task indicates successful completion of the offspring task or identifies specific error conditions.

### 6.1 DIRECTIVE SUMMARY

This section summarizes the directives for parent/offspring tasking and inter-task communication.

#### 6.1.1 Parent/Offspring Tasking Directives

There are two classes of parent/offspring tasking directives:

1. *Spawning*—Directives that create a connection between tasks
2. *Chaining*—Directives that transfer a connection

Three directives can connect a parent task to an offspring task:

1. *Spawn*—This directive requests activation of, and connects to, a specific offspring task.

An offspring task spawned by a parent task can return current status information or exit status information to a connected parent task.

Spawn directive options include:

Queuing a command line for the offspring task

Establishing the offspring task's TI: (terminal)

For privileged tasks, designating any terminal as the offspring TI:

2. *Connect*—This directive establishes task communications for synchronizing with the exit status or emit status issued by a task that is already active.
3. *Send, Request, and Connect*—This directive sends data to the specified task, requests activation of the task if it is not already active, and connects to the task.

Two directives support task chaining:

1. *Request and Pass Offspring Information*—This directive allows an offspring task to pass its parent connection to another task, thus making the new task the offspring of the original parent. The RPOI\$ directive offers all the options of the Spawn directive.
2. *Send Data, Request and Pass Offspring Control Block*—This directive sends a data packet for a specified task, passes its parent connection to that task, and requests the task if it is not already active.

A parent task can use the Spawn and Connect directives to connect to more than one offspring task. In addition, the parent task can use the directives in any combination to multiply connect to offspring tasks.

An offspring task can be connected to multiple parent tasks. An appropriate data structure, the Offspring Control Block (OCB), is produced (in addition to those already present) each time a parent task connects to the offspring task.

### 6.1.2 Task Communication Directives

Two directives in an offspring task return status to connected parent tasks:

1. *Exit With Status*—This directive in an offspring task causes the offspring task to exit, passing status words to all connected parent tasks connected by a Spawn, Connect, or Send, Request, and Connect directive.
2. *Emit Status*—This directive causes the offspring task to pass status words to either the specified connected task or all connected parent tasks if no task is explicitly specified.

When status is passed to tasks in this manner, the parent task no longer remains connected.

Standard offspring task status values that can be returned to parent tasks are listed as follows:

EX\$WAR	0	Warning Task succeeded, but irregularities are possible
EX\$SUC	1	Success Results should be as expected
EX\$ERR	2	Error Results are unlikely to be as expected
EX\$SEV	4	Severe Error One or more fatal errors detected, or task aborted

These symbols are defined in DIRSYM.MAC. They become defined locally when the EXST\$ macro is invoked. However, the exit status may be any 16-bit value.

## 6.2 CONNECTING AND PASSING STATUS

Offspring task exit status can be returned to connected (parent) task by issuing the Exit With Status directive. Offspring tasks can return status to one or more connected parent tasks at any time by issuing the Emit Status Directive. Note that only connected parent-offspring tasks can pass status.

The means by which a task connects to another task are indistinguishable once the connect process is complete. For example, Task A can become connected to Task B in one of the following ways:

- Task A spawned Task B when Task B was inactive.
- Task A connected to Task B when Task B was active.
- Task A issued a Send, Request, And Connect directive to Task B when Task B was either active or inactive.
- Task A either spawned or connected to Task C, which then chained to Task B by means of either an RPOI\$ directive or an SDRP\$ directive.

Regardless of the way in which Task A became connected to Task B, Task B can pass status information back to Task A, set the event flag specified by Task A, or cause the AST specified by Task A to occur in any of the following ways. Note that once offspring task status is returned to one or more parent tasks, the parent tasks become disconnected.

- Task B issues a successful exit directive. Task A receives a status of EX\$SUC.
- Task B is aborted. Task A receives a severe error status of EX\$SEV.
- Task B issues an Exit With Status directive and return status to Task A upon completion of Task B.
- Task B issues an Emit Status directive specifying Task A. If Task A is multiply connected to Task B, the OCBs that contain information about these multiple connections are stored in a FIFO queue. The first OCB is used to determine which event flag, AST address, and exit status block to use.

- Task B issues an Emit Status directive to all connected tasks (no task name specified).

If a task specifies another task in a Spawn, Connect, or Send, Request, and Connect directive and then exits, and if status is not yet returned, the OCB representing this connect remains queued. However, the OCB is marked to indicate that the parent task has exited. When this OCB is subsequently dequeued by an Emit Status directive, or any type of exit, no action is taken since the parent task has exited. This procedure is followed to help a multiply connected task to remain synchronized when parent tasks unexpectedly exit.

Examples of using directives for intertask synchronization are provided (macro call form for directives are shown) in Table 6-1. Task A is the parent task and Task B is the offspring task.

Table 6-1  
Directive Examples for Intertask Synchronization

<i>Task A</i>	<i>Task B</i>	<i>Action</i>
SPWN\$	EXST\$	Task A spawns Task B. Upon Task B completion, Task B returns status to Task A.
CNCT\$	EXST\$	Task A connects to active Task B. Upon Task B completion, Task B returns status to Task A.
SDRC\$	RCVX\$, EMST\$	Task A sends data to Task B, requests Task B if it is not active, and connects to Task B. Task B receives the data, does some processing based on the data, returns status to Task A (possibly setting an event flag or declaring an AST), and becomes disconnected from Task A.
\$SDRC\$, USTP\$	RCST\$, EMST\$	Task A sends data to Task B, requests Task B if it is not active, connects to Task B, and unstops Task B (if Task B previously could not dequeue the data packet). Task B receives the data, does some processing based on the data, and returns status to Task A (possibly setting an event flag or declaring an AST).
SDAT\$, USTP\$	RCST\$	Task A queues a data packet for Task B and unstops Task B; Task B receives the data.
SPWN\$	RPOI\$, SDRP\$	Task A spawns Task B. Task B chains to Task C by issuing an RPOI\$ or an SDRP\$ directive. Task A is now Task C's parent. Task A is no longer connected to Task B.



# CHAPTER 7

## MEMORY MANAGEMENT DIRECTIVES

---

Within the framework of memory management directives, this chapter discusses the concepts of extended logical address space, regions, and virtual address windows.

### 7.1 ADDRESSING CAPABILITY OF A SYSTEM TASK

Without overlays, a task cannot explicitly refer to a location with an address greater than 177777 (32K words). The 16-bit word size imposes this restriction on a task's addressing capability. Overlaying a task means that it must first be divided into segments: a single root segment, which is always in memory; and any number of overlay segments, which can be loaded into memory as required. Unless a task uses the memory management directives described in this chapter, the combined size of the task segments concurrently in memory cannot exceed 32K words.

When resident task segments cannot exceed a total of 32K words, a task requiring large amounts of data must access that data on disk. Data is disk-based not only because of limited memory space but also because transmission of large amounts of data between tasks is only practical by means of disk. An overlaid task, or a task that needs to access or transfer large amounts of data, incurs a considerable amount of transfer activity over that caused by the task's function.

Task execution could obviously be faster if all or a greater portion of the task were resident in memory at run time. The system includes a group of memory management directives that provide the task with this capability. The directives overcome the 32K-word addressing restriction by allowing the task to dynamically change the physical locations that are referred to by a given range of addresses. With these directives, a task can increase its execution speed by reducing its disk I/O requirements at the expense of increased physical memory requirements.

### 7.1.1 Address Mapping

In a mapped system, you do not need to know where a task resides in physical memory. Mapping, the process of associating task addresses with available physical memory, is transparent to the user and is accomplished by memory management hardware. When a task references a location (virtual address), the memory management hardware determines the physical address in memory. The memory management directives use the hardware to perform address mapping at a level that you can see and control.

### 7.1.2 Virtual and Logical Address Space

The three concepts defined here (physical address space, logical address space, and virtual address space) provide a basis for understanding the functions performed by the memory management directives:

- Physical Address Space*—A task's physical address space is the entire set of physical memory addresses.
- Logical Address Space*—A task's logical address space is the total amount of physical memory to which the task has access rights. This includes various areas called regions (see Section 7.3). Each region occupies a contiguous block of memory.
- Virtual Address Space*—A task's virtual address space corresponds to the 32K-word address range imposed by the 16-bit word length. The task can divide its virtual address space into segments called virtual address windows (see Section 7.2).

If the capabilities supplied by the memory management directives were not available, a task's virtual address space and logical address space would directly correspond; a single virtual address would always point to the same logical location. Both types of address space would have a maximum size of 32K words. However, the ability of the memory management directives to assign or map a range of virtual addresses (a window) to different logical areas (regions) enables you to extend a task's logical address space beyond 32K words.

## 7.2 VIRTUAL ADDRESS WINDOWS

To manipulate the mapping of virtual addresses to various logical areas, you must first divide a task's 32K of virtual address space into segments. These segments are called virtual address windows. Each window encompasses a contiguous range of virtual addresses, which must begin on a 4K-word boundary (that is, the first address must be a multiple of 4K). The number of windows defined by a task can vary from 1 through 23. For all tasks, window 0 is not available to you. The size of each window can range from a minimum of 32 words through a maximum of 32K words.

A task that includes directives to manipulate address windows dynamically must have window blocks set up in its task header. The Executive uses window blocks to identify and describe each currently existing window. You can specify the required number of additional window blocks—that is, the number of windows created by the memory management directives—to be set up by the Task Builder. (See the *RSX-11M/M-PLUS Task Builder Reference Manual*.) The number of blocks that you specify should equal the maximum number of windows that will exist at any one time when the task is running.

A window's identification is a number from 0 through  $15_{10}$  for user windows; it is an index to the window's corresponding window block. The address window identified by 0 is the window that maps the task's header and root segment. The Task Builder automatically creates window 0, which is mapped by the Executive and cannot be specified in any directive.

Figure 7-1 shows the virtual address space of a task divided into four address windows (windows 0, 1, 2, and 3). The shaded areas indicate portions of the address space that are not included in any window (9K through 12K and 23K through 24K). Addresses that fall within the ranges corresponding to the shaded areas cannot be used.

When a task uses memory management directives, the Executive views the relationship between the task's virtual and logical address space in terms of windows and regions. Unless a virtual address is part of an existing address window, reference to that address will cause an illegal address trap to occur. Similarly, a window can be mapped only to an area that is all or part of an existing region within the task's logical address space (see Section 7.3).

Once a task has defined the necessary windows and regions, it can issue memory management directives to perform operations such as the following:

- Map a window to all or part of a region
- Unmap a window from one region to map it to another region
- Unmap a window from one part of a region in order to map it to another part of the same region

### 7.3 REGIONS

A region is a portion of physical memory to which a task has (or potentially may have) access. The current window-to-region mapping context determines that part of a task's logical address space that the task can access at one time. A task's logical address space can consist of various types of regions:

- Task region*—A contiguous block of memory in which the task runs
- Static common region*—An area, such as a global common area.
- Dynamic region*—A region created dynamically at run time by issuing the memory management directives
- Shareable region*—A read-only portion of multiuser tasks that are in shareable regions

Note: Static common regions are dynamically loaded whenever needed.

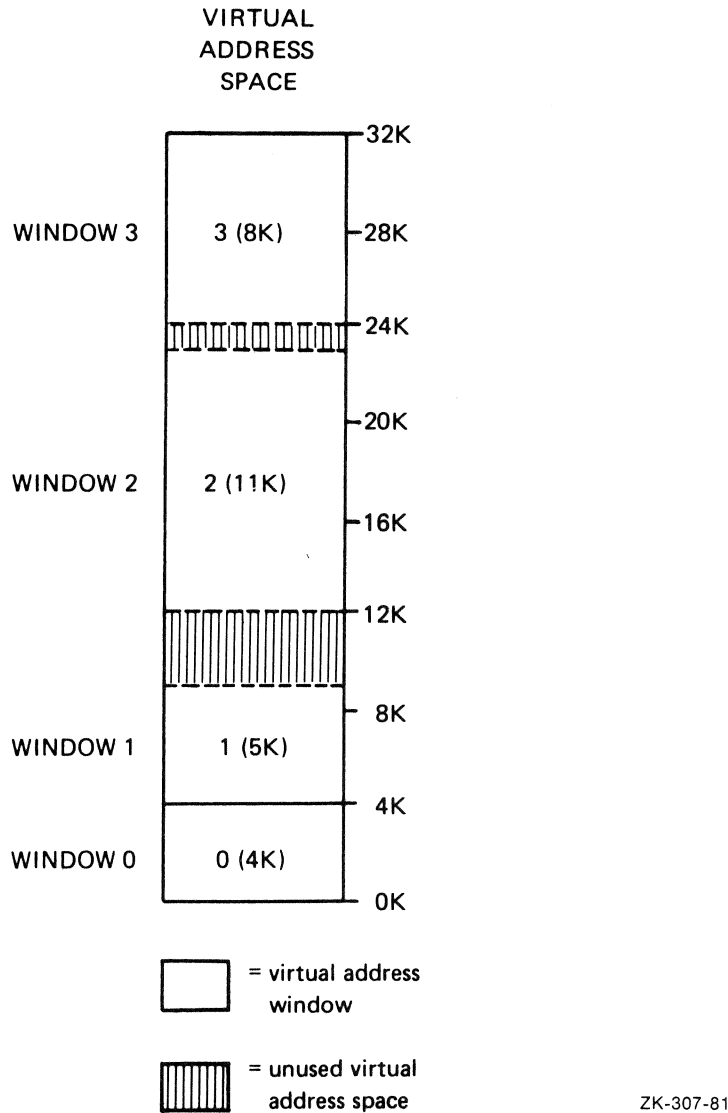
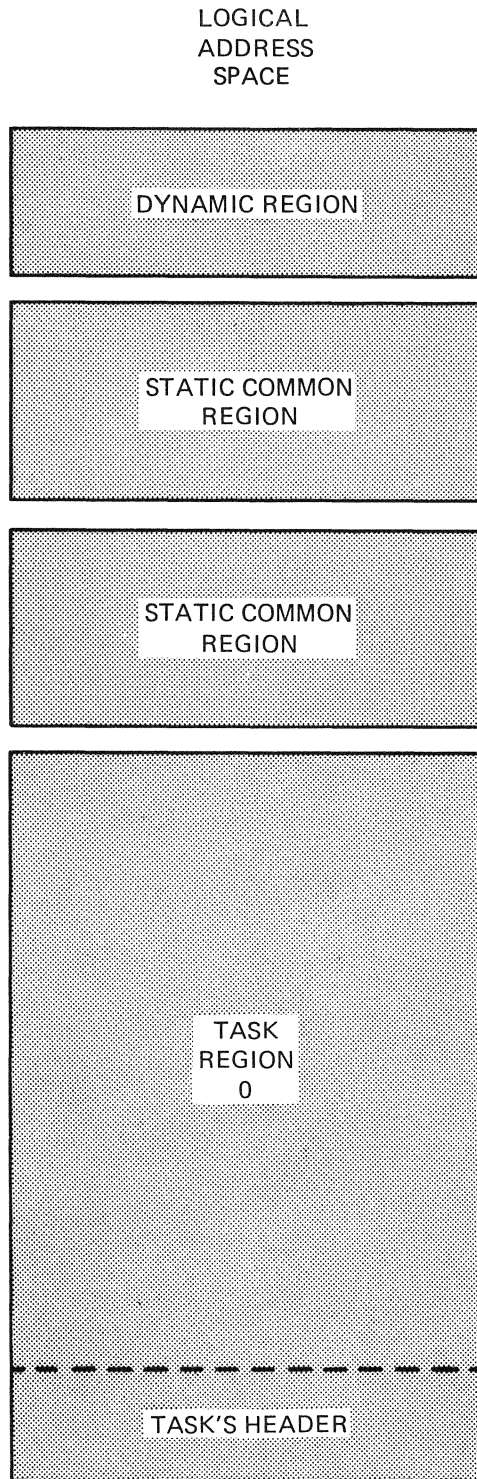


Figure 7-1  
Virtual Address Windows

Tasks refer to a region by means of a region ID returned to the task by the Executive. A region ID from 0 through 23 refers to a task's static attachment. Region ID 0 always refers to a task's task region. Region ID 1 always refers to the read-only (pure code) portion of multiuser tasks. All other region IDs are actually addresses of the attachment descriptor maintained by the Executive in the system dynamic storage area.

Figure 7-2 shows a sample collection of regions that could make up a task's logical address space at some given time. The header and root segment are always part of the task region. Since a region occupies a contiguous area of memory, each region is shown as a separate block.

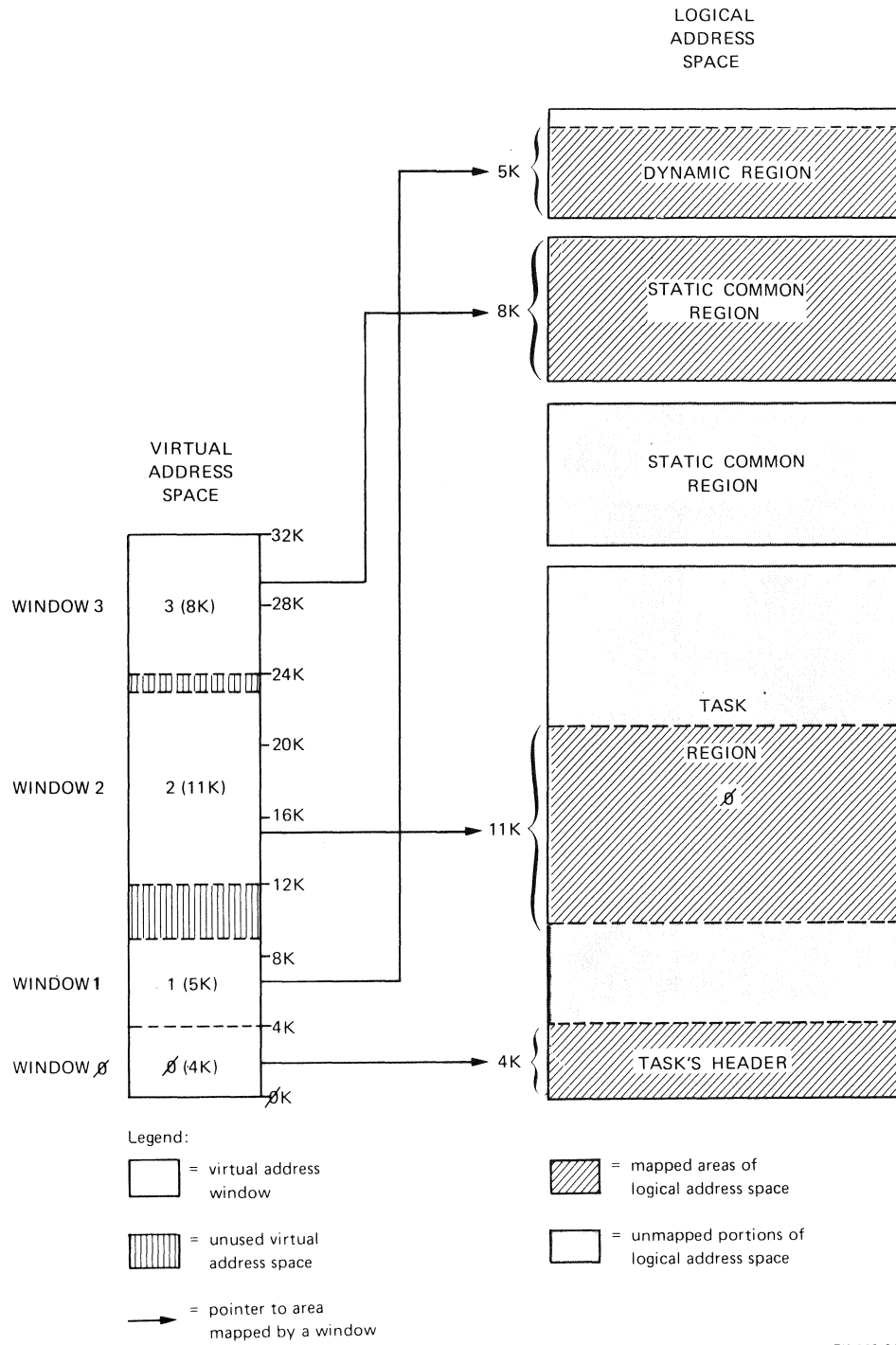
Figure 7-3 illustrates a possible mapping relationship between the windows and regions shown in Figures 7-1 and 7-2.



ZK-308-81

Figure 7-2  
Region Definition Block

7-6 MEMORY MANAGEMENT DIRECTIVES



ZK-309-81

Figure 7-3  
Mapping Windows to Regions

### 7.3.1 Shared Regions

Address mapping not only extends a task's logical address space beyond 32K words, it also allows the space to extend to regions that have not been linked to the task at task-build time. One result is an increased potential for task interaction by means of shared regions. For example, a task can create a dynamic region to accommodate large amounts of data. Any number of tasks can then access that data by mapping to the region. Another result is the ability of tasks to use a greater number of common routines. Thus, tasks can map to required routines at run time, rather than linking to them at task-build time.

### 7.3.2 Attaching to Regions

Attaching is the process by which a region becomes part of a task's logical address space. A task can map only a region that is part of the task's logical address space. There are three ways to attach a task to a region:

1. All tasks are automatically attached to regions that are linked to them at task-build time.
2. A task can issue a directive to attach itself to a named static common region or a named dynamic region.
3. A task can request the Executive to attach another specified task to any region within the logical address space of the requesting task.

Attaching identifies a task as a user of a region and prevents the system from deleting a region until all user tasks have been detached from it. (It should be noted that fixed tasks do not automatically become detached from regions upon exiting.)

Note: Each Send By Reference directive issued by a sending task creates a new attachment descriptor for the receiving task. However, multiple Send By Reference directives referencing the same region require only one attachment descriptor. After the receiving task issues a series of Receive By Reference directives and receives all pending data requests, the task should detach the region to return the attachment descriptors to the pool.

You can avoid multiple attachment descriptors when sending and receiving data by reference. Setting the WS.NAT bit in the Window Definition Block (see Section 7.5.2) causes the Executive to create a new attachment descriptor for that region only if necessary (that is, if the task is currently not attached to the region).

### 7.3.3 Region Protection

A task cannot indiscriminately attach to any region. Each region has a protection mask to prevent unauthorized access. The mask indicates the types of access (read, write, extend, delete) allowed for each category of user (system, owner, group, world). The Executive checks that the requesting task's User Identification Code (UIC) allows it to make the attempted access. The attempt fails if the protection mask denies that task the access it wants.

To determine when tasks may add to their logical address space by attaching regions, the following points must be considered (note that all considerations presume there is no protection violation):

- Any task can attach to a named dynamic region, provided the task knows the name. In the case of an unnamed dynamic region, a task can attach to the region only after receiving a Send By Reference directive from the task that created the region.
- Any task can issue a Send By Reference directive to attach another task to any region. The task region itself may not be one of the regions involved. The reference sent includes the access rights with which the receiving task attaches to the region. The sending task can only grant access rights that it has itself.
- Any task can map to a named static common region.

## 7.4 DIRECTIVE SUMMARY

This section briefly describes the function of each memory management directive. Chapter 9 defines all the directives in detail.

### 7.4.1 Create Region Directive (CRRG\$)

The Create Region directive creates a dynamic region in a designated system-controlled partition and optionally attaches the issuing task to it.

### 7.4.2 Attach Region Directive (ATRG\$)

The Attach Region directive attaches the issuing task to a static common region or to a named dynamic region.

### 7.4.3 Detach Region Directive (DTRG\$)

The Detach Region directive detaches the issuing task from a specified region. Any of the task's address windows that are mapped to the region are automatically unmapped.

### 7.4.4 Create Address Window Directive (CRAW\$)

The Create Address Window directive creates an address window, establishes its virtual address base and size, and optionally maps the window. Any other windows that overlap with the range of addresses for the new window are first unmapped and then eliminated.

### 7.4.5 Eliminate Address Window Directive (ELAW\$)

The Eliminate Address Window directive eliminates an existing address window, unmapping it first if necessary.



#### **7.4.6 Map Address Window Directive (MAP\$)**

The Map Address Window directive maps an existing window to an attached region. The mapping begins at a specified offset from the start of the region and goes to a specified length. If the window is already mapped elsewhere, the Executive unmaps it before carrying out the map assignment described in the directive.

#### **7.4.7 Unmap Address Window Directive (UMAP\$)**

The Unmap Address Window directive unmaps a specified window. After the window is unmapped, its virtual address range cannot be referenced until the task issues another mapping directive.

#### **7.4.8 Send By Reference Directive (SREF\$)**

The Send By Reference directive inserts a packet containing a reference to a region into the receive queue of a specified task. The receiver task is automatically attached to the region referred to.

#### **7.4.9 Receive By Reference Directive (RREF\$)**

The Receive By Reference directive requests the Executive first to select the next packet from the receive-by-reference queue of the issuing task, and then to make the information in the packet available to the task. Optionally the directive can map a window to the referenced region or cause the task to exit if the queue does not contain a receive-by-reference packet.

#### **7.4.10 Get Mapping Context Directive (GMCX\$)**

The Get Mapping Context directive causes the Executive to return to the issuing task a description of the current window-to-region mapping assignments. The description is in a form that enables the user to restore the mapping context through a series of Create Address Window directives.

#### **7.4.11 Get Region Parameters Directive (GREG\$)**

The Get Region Parameters directive causes the Executive to supply the issuing task with information about either its task region (if no region ID is given) or an explicitly specified region.

### **7.5 USER DATA STRUCTURES**

Most memory management directives are individually capable of performing a number of separate actions. For example, a single Create Address Window directive can unmap and eliminate as many as seven conflicting address windows, create a new window, and map the new window to a specified region.

The complexity of the directives requires a special means of communication between the user task and the Executive. The communication is achieved through data structures that:

- Allow the task to specify which directive options it wants the Executive to perform
- Permit the Executive to provide the task with details about the outcome of requested actions

There are two types of user data structures that correspond to the two key elements (regions and address windows) manipulated by the directives. The structures are called:

The Region Definition Block (RDB)

The Window Definition Block (WDB)

Every memory management directive, except Get Region Parameters, uses one of these structures as its communications area between the task and the Executive. Each directive issued includes in the DPB a pointer to the appropriate definition block. The task assigns symbolic address offset values that point to locations within an RDB or a WDB. The task can change the contents of these locations to define or modify the directive operation. After the Executive has carried out the specified operation, it assigns values to various locations within the block to describe the actions taken and to provide the task with information useful for subsequent operations.

### 7.5.1 Region Definition Block (RDB)

Figure 7-4 illustrates the format of an RDB. In addition to the symbolic offsets defined in the diagram, the region status word R.GSTS contains defined bits that may be set or cleared by the Executive or the task. Table 7-1 shows the bits and their definitions.

These symbols are defined by the RDBDF\$ macro, as described in Section 7.5.1.1.

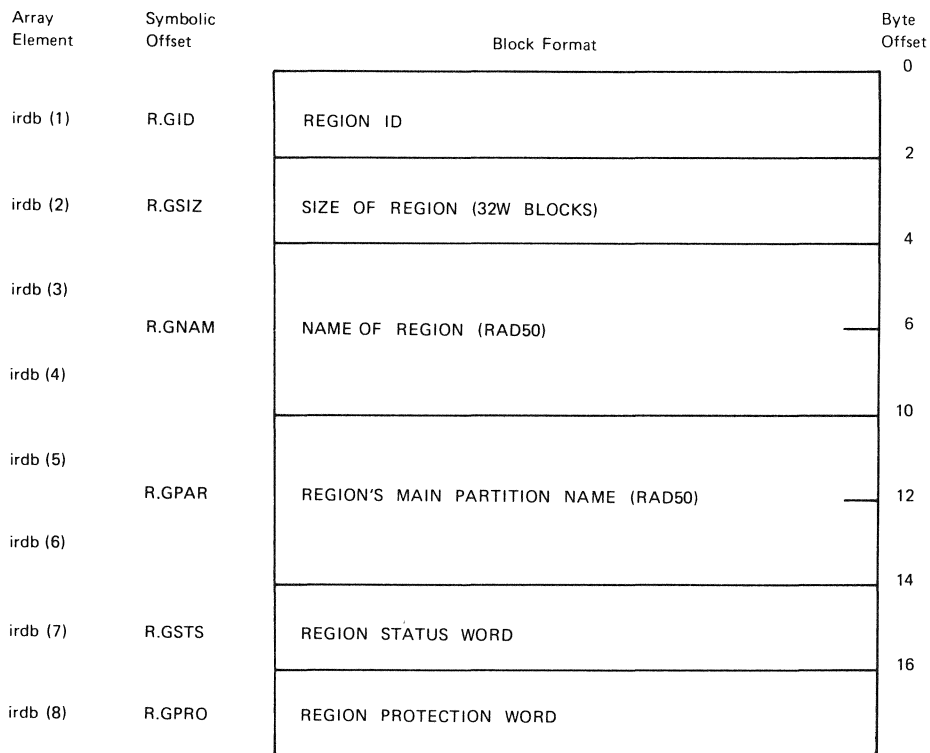
The three memory management directives that require a pointer to an RDB are:

Create Region (CRRG\$)

Attach Region (ATRG\$)

Detach Region (DTRG\$)

When a task issues one of these directives, the Executive clears the four high-order bits in the region status word of the appropriate RDB. After completing the directive operation, the Executive sets the RS.CRR or RS.UNM bit to indicate to the task what actions were taken. The Executive never modifies the other bits.



ZK-310-81

Figure 7-4  
Region Definition Block

Table 7-1  
Bits of the Region Status Word

<i>Bits</i>	<i>Definition</i>
RS.CRR=100000	Region was successfully created.
RS.UNM=40000	At least one window was unmapped on a detach.
RS.MDL=200	Mark region for deletion on last detach. When a region is created by a CRRG\$ directive, the region is normally marked for deletion on last detach. However, if RS.NDL is set when the CRRG\$ directive is executed, the region is not marked for deletion. Subsequent execution of a DTRG\$ directive with RS.MDL set marks the region for deletion.
RS.NDL=100	Created region is not to be marked for deletion on last detach.
RS.ATT=40	Attach to created region.
RS.NEX=20	Created region is not extendable.
RS.DEL=10	Delete access desired on attach.
RS.EXT=4	Extend access desired on attach.
RS.WRT=2	Write access desired on attach.
RS.RED=1	Read access desired on attach.

**7.5.1.1 Using Macros to Generate an RDB** —The system provides two macros, RDBDF\$ and RDBBK\$, to generate and define an RDB. RDBDF\$ defines the offsets and status word bits for a region definition block; RDBBK\$ then creates the actual region definition block.

The format of RDBDF\$ is:

RDBDF\$

Since RDBBK\$ automatically invokes RDBDF\$, you need specify only RDBBK\$ in a module that creates an RDB. The format of the call to RDBBK\$ is:

RDBBK\$ siz,nam,par,sts,pro

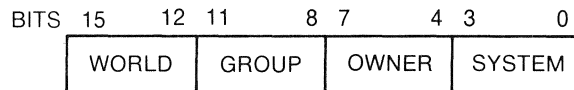
- siz The region size in 32-word blocks.
- nam The region name (RAD50).
- par The name of the partition in which to create the region (RAD50).
- sts Bit definitions of the region status word.
- pro The region's default protection word.

The sts argument sets specified bits in the status word R.GSTS. The argument normally has the following format:

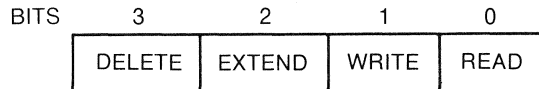
<bit1[! ... !bitn]>

- bit A defined bit to be set.

The argument pro is an octal number. The 16-bit binary equivalent specifies the region's default protection as follows:



Each of these four categories has four bits, with each bit representing a type of access:



A bit value of 0 indicates that the specified type of access is to be allowed; a bit value of 1 indicates that the specified type of access is to be denied.

The macro call

```
RDBBK$
102.,ALPHA,GEN,<RS.NDL!RS.ATT!RS.WRT!RS.RED>,167000
```

expands to:

```
.WORD      0
.WORD      102.
.RAD50     /ALPHA/
.RAD50     /GEN/
.WORD      0
.WORD      RS.NDL!RS.ATT!RS.WRT!RS.RED
.WORD      167000
```

If a Create Region directive pointed to the RDB defined by this expanded macro call, the Executive would create a region 102<sub>10</sub> 32-word blocks in length, named ALPHA, in a partition named GEN. The defined bits specified in the sts argument tell the Executive:

- Not to mark the region for deletion on the last detach
- To attach region ALPHA to the task issuing the directive macro call
- To grant read and write access to the attached task

The protection word specified as 167000<sub>8</sub> assigns a default protection mask to the region. The octal number, which has a binary equivalent of 1110 1110 0000 0000, grants access as follows:

System (1110)	All access
Owner (1110)	All access
Group (0000)	Read access only
World (0000)	Read access only

If the Create Region directive is successful, the Executive will first return to the issuing task a region ID value in the location accessed by symbolic offset R.GID, and then will set the defined bit RS.CRR in the status word R.GSTS.

**7.5.1.2 Using Fortran to Generate an RDB** —When programming in Fortran, you must create an 8-word, single-precision integer array as the RDB to be supplied in the subroutine calls:

CALL ATRG	(Attach Region directive)
CALL CRRG	(Create Region directive)
CALL DTRG	(Detach Region directive)

(See the *PDP-11 FORTRAN-77 Language Reference Manual* for information on the creation of arrays.)

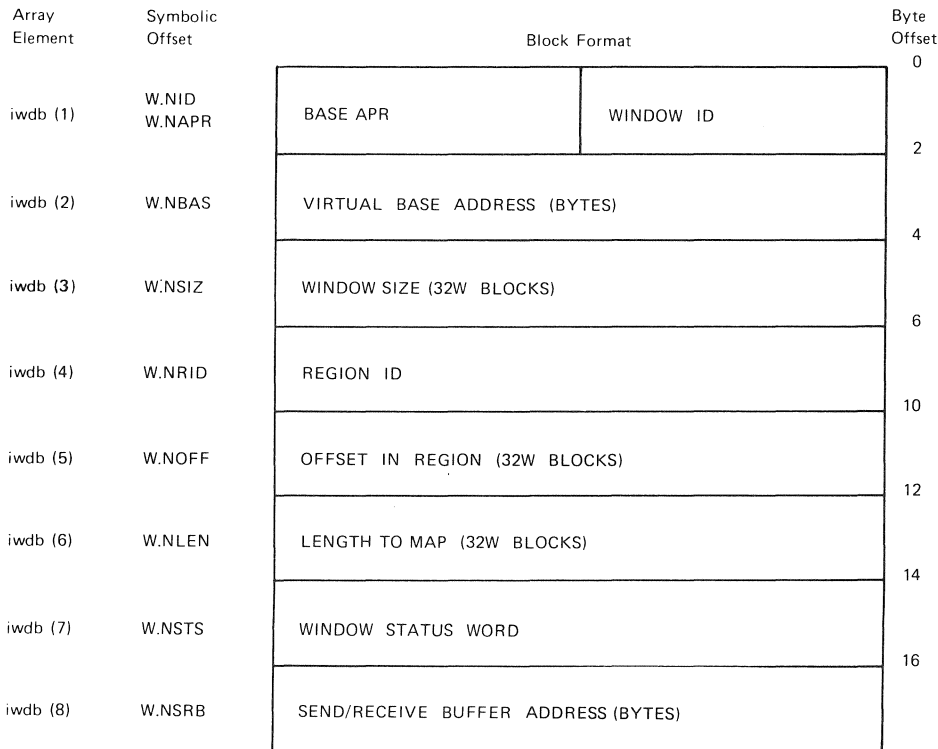
Table 7-2 shows the RDB array format.

Table 7-2  
RDB Array Format

<i>Word</i>	<i>Comment</i>
irdb(1)	Region ID
irdb(2)	Size of the region in 32-word blocks
irdb(3), irdb(4)	Region name (2 words in Radix-50 format)
irdb(5), irdb(6)	Name of the partition that contains the region (2 words in Radix-50 format)
irdb(7)	Region status word
irdb(8)	Region protection code

You can modify the region status word irdb(7) by setting or clearing the appropriate bits. See the list in Section 7.5.1 that describes the defined bits. The bit values are listed beside the symbolic offsets.

Note that Hollerith text strings can be converted to Radix-50 values by calls to The Fortran library routine IRAD50 (see the appropriate Fortran user's guide).



ZK-311-81

Figure 7-5  
Window Definition Block

### 7.5.2 Window Definition Block (WDB)

Figure 7-5 illustrates the format of a WDB. The block consists of a number of symbolic address offsets to specific WDB locations. One of the locations is the window status word W.NSTS, which contains defined bits that can be set or cleared by the Executive or the task. The bits and their definitions are shown in Table 7-3.

Table 7-3  
WDB Format

<i>Bit</i>	<i>Definition</i>
WS.CRW=100000	Address window was successfully created.
WS.UNM=40000	At least one window was unmapped by a Create Address Window, Map Address Window, or Unmap Address Window directive.
WS.ELW=20000	At least one window was eliminated in a Create Address Window or Eliminate Address Window directive.
WS.RRF=10000	Reference was successfully received.
WS.NBP=4000	Do not bypass cache for CRAW\$ directives.
WS.BPS=4000	Always bypass cache for MAP\$ directives.
WS.RES=2000	Map only if resident.
WS.NAT=1000	Create attachment descriptor only if necessary (for Send By Reference directives).
WS.64B=400	Defines the task's permitted alignment boundaries—0 for 256-word (512-byte) alignment, 1 for 32-word (64-byte) alignment.
WS.MAP=200	Window is to be mapped in a Create Address Window or Receive By Reference directive.
WS.RCX=100	Exit if no references to receive.
WS.DEL=10	Send with delete access.
WS.EXT=4	Send with extend access.
WS.WRT=2	Send with write access or map with write access.
WS.RED=1	Send with read access. These symbols are defined by the WDBDF\$ macro, as described in Section 7.5.2.1.

The following directives require a pointer to a WDB:

- Create Address Window (CRAW\$)
- Eliminate Address Window (ELAW\$)
- Map Address Window (MAP\$)
- Unmap Address Window (UMAP\$)
- Send By Reference (SREF\$)
- Receive By Reference (RREF\$)

When a task issues one of these directives, the Executive clears the four high-order bits in the window status word of the appropriate WDB. After completing the directive operation, the Executive can then set any of these bits to tell the task what actions were taken. The Executive never modifies the other bits.

**7.5.2.1 Using Macros to Generate a WDB** —The system provides two macros, WDBDF\$ and WDBBK\$, to generate and define a WDB. WDBDF\$ defines the offsets and status word bits for a window definition block; WDBBK\$ then creates the actual window definition block.

The format of WDBDF\$ is:

WDBDF\$

Since WDBBK\$ automatically invokes WDBDF\$, you need specify only WDBBK\$ in a module that generates a WDB. The format of the call to WDBBK\$ is:

WDBBK\$ apr,siz,rid,off,len,sts,srb

- apr A number from 0 through 7 that specifies the window's base Active Page Register (APR). The APR determines the 4K boundary on which the window is to begin. APR 0 corresponds to virtual address 0, APR 1 to 4K, APR 2 to 8K, and so on.
- siz The size of the window in 32-word blocks.
- rid A region ID.
- off The offset (in 32-word blocks) within the region to be mapped.
- len The length (in 32-word blocks) within the region to be mapped (defaults to the value of siz).
- sts The bit definitions of the window status word.
- srb A send/receive buffer virtual address.

The argument sts sets specified bits in the status word W.NSTS. The argument normally has the following format:

<bit1[! ... !bitn]>

- bit A defined bit to be set.



The macro call

```
WDBBK$ 5,76.,0,50.,,<WS.64B!WS.MAP!WS.WRT>
```

expands to:

```
.BYTE    0,5      (Window ID returned in low-order byte)
.WORD    0        (Base virtual address returned here)
.WORD    76.
.WORD    0
.WORD    50.
.WORD    0
.WORD    WS.64B!WS.MAP!WS.WRT
.WORD    0
```

If a Create Address Window directive pointed to the WDB defined by the expanded macro call, the Executive would:

- Create a window  $76_{10}$  blocks long beginning at APR 5—virtual address 20K or  $120000_8$
- Map the window with write access (<WS.MAP!WS.WRT>) to the issuing task's task region (because the macro call specified 0 for the region ID)
- Start the map  $50_{10}$  blocks from the base of the region, and map an area either equal to the length of the window— $76_{10}$ —or to the length remaining in the region, whichever is smaller (because the macro call defaulted the len argument) and align the window on a 64-byte boundary.
- Return values to the symbolic W.NID (the window's ID) and W.NBAS (the window's virtual base address)

**7.5.2.2 Using Fortran to Generate a WDB** —You must create an 8-word, single-precision integer array as the WDB to be supplied in the subroutine calls:

```
CALL CRAW      (Create Address Window directive)
CALL ELAW     (Eliminate Address Window directive)
CALL MAP      (Map Address Window directive)
CALL UNMAP    (Unmap Address Window directive)
CALL SREF     (Send By Reference directive)
CALL RREF     (Receive By Reference directive)
```

(See the *PDP-11 FORTRAN-77 Language Reference Manual* for information on the creation of arrays.)

Table 7-4 shows the WDB array format.

Table 7-4  
WDB Array Format

<i>Word</i>	<i>Contents</i>
iwdb(1)	Bits 0 through 7 contain the window ID; bits 8 through 15 contain the window's base APR.
iwdb(2)	Base virtual address of the window.
iwdb(3)	Size of the window in 32-word blocks.
iwdb(4)	Region ID.
iwdb(5)	Offset length (in 32-word blocks) within the region at which map begins.
iwdb(6)	Length (in 32-word blocks) mapped within the region.
iwdb(7)	Window status word.
iwdb(8)	Address of send/receive buffer.

You can modify the window status word iwdb(7) by setting or clearing the appropriate bits. See the list in Section 7.5.2 that describes the defined bits. The bit values are listed alongside the symbolic offsets.

The contents of bits 8 through 15 of iwdb(1) must normally be set without destroying the value in bits 0 through 7 for any directive other than the Create Address Window.

A call to GETADR (see Section 3.4.1.4) can be used to set up the address of the send/receive buffer. For example:

```
CALL GETADR(IWDB,,,,,,,,,IRCVB)
```

This call places the address of buffer IRCVB in array element 8. The remaining elements are unchanged. The subroutines SREF and RREF also set this value. If you use the SREF and RREF routines, you do not need to use GETADR.

### 7.5.3 Assigned Values or Settings

The exact values or settings assigned to individual fields within the RDB or the WDB vary according to each directive. Fields that are not required as input can have any value when the directive is issued. Chapter 6 describes which offsets and settings are relevant for each memory management directive. The values assigned by the task are called input parameters, whereas those assigned by the Executive are called output parameters.

## 7.6 PRIVILEGED TASKS

When a privileged task maps to the Executive and the I/O page, the system normally dedicates five or six APRs to this mapping. A privileged task can issue memory management directives to remap any number of these APRs to regions. Take great care when using the directives in this way, because such remapping can cause obscure bugs to occur. When a directive unmaps a window that formerly mapped the Executive or the I/O page, the Executive restores the former mapping.

Note: Tasks should not remap APR0. Remapping APR0 causes information such as the DSW, overlay structures, or language runtime systems to become inaccessible.



## CHAPTER 8

# CALLABLE SYSTEM ROUTINES

---

The system provides a set of callable routines that are invoked by the PDP-11 standard R5 calling sequence. The routines themselves are provided in a resident library called POSSUM, against which you must task build your programs. A program calls a routine in the POSSUM library to have a specified service executed. Some of the routines use a separate task in the system called a server. This chapter describes each callable routine as well as the name of any server that a particular routine may require.

POSSUM can be included as part of a cluster of libraries with RMSRES and other libraries. See the *RSX-11M/M-PLUS Task Builder* manual for details on cluster libraries.

Note: When you link programs to run on the Professional, invoke the Task Builder using the name PAB (Professional Application Builder) rather than TKB.

You can provide one of two options in your task build command file to include the POSSUM library in your task:

Use the following Task Builder format to link a task to the POSSUM resident library:

```
LIBR=POSSUM:RO
```

Use this Task Builder format to link a task to a cluster library which includes the POSSUM resident library:

```
CLSTR=POSSUM,OTHER:RO
```

### 8.1 GENERAL CONVENTIONS FOR ALL CALLABLE SYSTEM ROUTINES

This section defines the general mechanism used for calling all the defined system routines in the POSSUM resident library.

### 8.1.1 PDP-11 R5 Calling Sequence

Your program must use register 5 (R5) to pass the address of an argument list that resides in your task's data space. The argument list itself is of variable length, so that only the necessary arguments are passed.

The general MACRO-11 coding sequence of the call follows.

Instruction space coding sequence:

```
MOV    #ARGLST,R5    ; address of the argument list to pass
JSR    PC,SUB        ; call the subroutine
```

Data space coding sequence:

```
ARGLST: .BYTE    NUMBER,0    ; NUMBER is the number of arguments
                                     ; following in the list
        .WORD    ADDR1      ; address of first argument
        ...
        .WORD    ADDRn     ; the nth argument
```

For higher level languages that support the R5 calling sequence (such as BASIC-PLUS-2 or FORTRAN-77), see your language reference manual or user guide for correct syntax. The examples in this chapter assume BASIC-PLUS-2 as the high level language being used. All examples assume a higher level language call.

In BASIC-PLUS-2, you can invoke the previous MACRO-11 call as follows:

```
120      CALL SUB BY REF (ADDR1%,...,ADDRn%)
```

BP2 internally formats an R5 calling block and issues the call to the system routine for you.

### 8.1.2 Conventions for Callable System Services

All of the routines documented in this chapter have specific conventions that you must follow for programming success:

- All arguments passed to the system routines are by reference. This means that you are passing the address of the value in your program to the routine in POSSUM.
- Every routine shares a common format in that the first argument is the address of an 8-word Status Control Block found in your program to which the routines return completion status. The Status Control Block is always eight words in length, so care must be taken to allocate the proper amount of space in your program.
- Every routine requires a request parameter. All of the routines are multi-purpose and this 1-word request parameter is the method for specifying which option(s) to execute.

- When specifying either a device or file name string as a required element in an argument list, always specify the accompanying size field in bytes. (A byte corresponds to one ASCII character.)
- The system services preserve registers R0-R4. This is of no concern to higher level language programmers since the languages preserve internal registers around the call.

### 8.1.3 Status Control Block Format

The 8-word Status Control Block has the following format:

Word 0	is the count of the number of status (error) parameters passed back to the Status Control Block upon completion of the routine
Word 1	is the overall call status. This is a 1-word value defined as follows: <ul style="list-style-type: none"> <li>+1 Success</li> <li>-1 Directive Status Error. The actual \$DSW error is in word 3.</li> <li>-2 A QIO error. The contents of the 2-word QIO status block are in words 3 and 4.</li> <li>-3 An RMS error. The RMS STS and STV fields are returned in words 3 and 4.</li> <li>-4 Server specific error. The contents of words 3 through 8 are defined by each routine.</li> <li>-5 Interface error. An error occurred when trying to interpret the argument block. Currently, one of the following values would be in word 3.           <ul style="list-style-type: none"> <li>-1 Feature not supported. The code is not yet complete to execute the documented feature.</li> <li>-2 Impure area is invalid, or missing. Usually indicates that you have not correctly taskbuilt your program.</li> <li>-3 Invalid number of parameters (too few or too many).</li> </ul> </li> </ul>
Word 2-7	is as defined above

## 8.2 CALLABLE TASK ROUTINES

The POSSUM resident library contains routines that perform specific functions. Those routines are:

- PROATR gets or sets file attributes
- PRODIR creates or deletes a directory
- PROFBI formats, initializes, and checks for bad blocks on disks

- PROLOG translates, creates, and deletes a logical name
- PROVOL mounts, dismounts, bootstraps, and/or writes the bootblock on a volume

The following sections describe each callable routine in detail.

### 8.3 PROATR

The PROATR routine provides a means of accessing certain file attributes. Given a file ID or a file specification and an attribute list, the GET function uses the attribute list to determine which attributes to read and where to store the associated information. Conversely, the SET function writes the attribute information specified in the attribute list to the file.

The PROATR routine provides two forms of accessing file attributes. You can use PROATR to:

- Get attributes of a file
- Set attributes of a file

The PROATR routine does not require a server to execute.

To get or set file attributes, invoke the PROATR routine with the following arguments:

STATUS, REQUEST, ATTRIBUTE\_LIST, FILE\_ID, LUN

where:

STATUS	The address of the 8-word Status Control Block
REQUEST	The address of a word containing the decimal value of the operation to be performed (see Section 8.3.1)
ATTRIBUTE_LIST	The address of the attribute list (see Section 8.3.1)
FILE_ID	The address of a buffer that contains a 3-word Files-11 FID
LUN	The address of a buffer that contains the LUN number used to obtain the file ID (see Section 8.3.1)

#### 8.3.1 How to Specify the REQUEST Argument in PROATR

PROATR uses the decimal value specified in the REQUEST argument to determine, first, whether to get or set file attributes and, second, whether the input file descriptor is a file ID or an ASCII file specification.

When specifying this argument, use the combined values of the desired function plus the input file descriptor or ASCII file specification as described below.



**Function**

- 0 Get file attributes
- 1 Set file attributes

**Input File Descriptor**

- 0 The contents of the buffer address specified in the FILEID argument is a 3-word Files-11 FID (file ID). In this case, the address specified in the LUN argument contains the specific LUN used to obtain the file ID.
- 2 The contents of the buffer address specified in the FILEID argument is an ASCII file name specification. In this case, the address specified in the LUN argument contains the size of the file.

**Notes**

1. The file identification block is a 3-word block containing the file number, the file sequence number, and a reserved word.

FID: File number  
 +2: File sequence number  
 +4: Reserved

2. The attribute list contains a variable number of entries terminated by an byte containing all zeroes. The maximum number of entries in the attribute list is six.

An entry in the attribute list has the following format:

.BYTE Attribute type, Attribute size  
 .WORD Pointer to the attribute buffer

Table 8-1 is a list of the accessible file attributes.

Table 8-1  
 Accessible File Attributes

<i>Attribute Code</i>	<i>Attribute Type</i>	<i>Size in Octal Bytes</i>
1	File owner	6
2	Protection	4
3	File characteristics	2
4	Record I/O area	40
5	File name, type, version number	12 6 File type 4
7	Version number	2
11	Statistics block	12
16	Placement control	16

Note: The file name contained in the header is not associated with the name in a directory entry except by convention. Therefore, you cannot use the file ID to get the file name as specified in the directory; the name that the ACP returns is the name contained in the header.

### 8.3.2 PRODIR

The PRODIR routine provides two forms of directory manipulation. You can use PRODIR to:

- Create a directory on a device
- Delete a directory on a device

The name of the server used to execute PRODIR is CREDEL. This server must be installed in your system to perform any of indicated services. Otherwise, PRODIR returns a directive error in the Status Control Block (see Section 8.1.3). To create or delete a directory, invoke the PRODIR routine with the following arguments:

STATUS, REQUEST, FILE\_NAME, FILE\_SIZE

where:

STATUS      The address of the 8-word Status Control Block

REQUEST     The address of a word containing the decimal value indicating which operation (CREATE or DELETE) to perform (see Section 8.3.3)

FILE\_NAME   The address of a buffer containing an ASCII device and directory specification

The device specification takes the form `ddn`:

where:

`dd`   the device name

`n`     the device unit number

The directory specification takes the one of the following forms:

`[ggg,mmm]` such as `[301,3]`  
or  
`[gggmmm]` such as `[301003]`  
or  
`[name]` such as `[WILEY]`

where:

`ggg`   group

`mmm`   member

FILE\_SIZE   The address of a byte value containing the length of the string in FILE\_NAME

The following is a sample BP2 call to PRODIR:

```
100 CALL PRODIR BY REF
  (STATUS%(),REQUEST%,DFILE$,LEN(DFILE$))
```

### 8.3.3 Using the REQUEST Argument for Creating or Deleting a Directory

PRODIR uses the value specified in the REQUEST argument to determine whether to create or delete a directory, as follows:

- 1 CREATE directory
- 2 DELETE directory

Example 8-1 shows how to access PRODIR from a BASIC-PLUS-2 program.

#### Example 8-1: How to Access PRODIR from a BASIC-PLUS-2 Program

```

10  ! PROGRAM TO CREATE/DELETE DIRECTORIES
20  DIM STATUS%(7), REQ$(2)
25  REQ$(1)='Create' \ REQ$(2)='Delete'
30  PRINT 'Create or Delete (C/D) :'; \ LINPUT #0,REQ$ &
\  REQUEST% = 1 \ IF LEFT$(REQ$,1) = 'C' THEN 40 ELSE &
IF LEFT$(REQ$,1)<>'D' THEN 20 ELSE REQUEST% = 2
40  PRINT 'Name of Directory to ';REQ$(REQUEST%);' :'; &
\  LINPUT #0,DFILE$
100 CALL PRODIR BY REF (STATUS%( ),REQUEST%,DFILE$,LEN(DFILE$))
110 FOR K=0 TO 7 \ PRINT 'STATUS';K,STATUS%(K) \ NEXT K
999 END

```

See the *BASIC-PLUS-2 Documentation Supplement* for a description of the command and overlay descriptor files for BASIC-PLUS-2 programs.

## 8.4 PROFBI

The PROFBI routine provides the mechanism for preparing media for use on the system. The PROFBI routine allows you to:

- Format a volume
- Check a volume for bad blocks
- Initialize a volume

To format or initialize a volume or check it for bad blocks invoke the PROFBI routine with the following arguments:

STATUS, REQUEST, DEVICE\_SPEC, DEVICE\_SIZE,  
ATTRIBUTE\_LIST, ATTRIBUTE\_SIZE

where:

STATUS	The address of the 8-word Status Control Block. The last six words of the Status Control Block contain the volume label when a volume is successfully initialized
--------	---

**REQUEST**            The address of a word containing the decimal value indicating the operation to be performed (see Section 8.4.1)

Note: When preparing the hard disk, specify the REQUEST code either for format or bad, but not for both. Either code will perform both functions on RD-50-type devices in the same operation.

**DEVICE\_SPEC**        The address of a buffer containing a character string which is the device specification of the volume to be formatted, initialized, or checked for bad blocks

**DEVICE\_SIZE**        The address of a word containing the length of the string in **DEVICE\_SPEC**

**ATTRIBUTE\_LIST**    The address of the attribute list. The attribute list is a buffer of legal attributes, predominantly intended for use by Macro programmers (see Notes). Legal attributes in PROFBI are:

- 1    Volume label
- 2    ACS buffer

**ATTRIBUTE\_SIZE**    The address of a word containing the total size of the attribute list

Note: The contents of the buffer for the **ATTRIBUTE\_LIST** argument are optional. That is, you must specify the argument but the buffer need not contain a volume label or an ACS specification.

#### 8.4.1 Using the REQUEST Argument in PROFBI

The PROFBI routine uses the decimal value specified in the **REQUEST** argument to determine which operation to perform. Specify in the **REQUEST** argument the value listed below that corresponds to the operation you desire:

- 1    Format a volume (only works for the hard disk)
- 2    Check a volume for bad blocks
- 4    Initialize a volume

#### Notes

1. The minimum length of **DEVICE\_SPEC** is four characters—the 3-character device mnemonic followed by a colon (such as DW1:). The device portion of **DEVICE\_SPEC** must end with a colon.

If you are initializing a volume, part of the device specification can be the volume label which may be up to 12 characters (in the form DW1:SPECTROSCOPY). You may also specify the volume label in the attribute list instead. If you specify the volume label in both the **DEVICE\_SPEC** argument and the **ATTRIBUTE\_LIST** argument, the **DEVICE\_SPEC** argument overrides the **ATTRIBUTE\_LIST** argument.

2. If you omit the volume label when initializing a volume, PROFBI creates a default volume label using the date and time the volume was initialized. The default volume label format is:

DDMMYYHHMMSS

3. DEVICE\_SPEC may also be a logical name string. The logical name string must end with a colon. The number of logical name translations cannot exceed eight. A ninth translation results in an error condition.
4. PROFBI requires the string supplied in the DEVICE\_SPEC and DEVICE\_SIZE arguments when initializing a volume or checking it for bad blocks. The DEVICE\_SPEC argument is necessary when formatting a volume.
5. The ATTRIBUTE\_LIST argument is the means of specifying optional parameters. The attribute list for PROFBI is simply a buffer of legal attributes. The high byte in the first word of the attribute list specifies the attribute type. The low byte specifies the size of the attribute list buffer in bytes.

You can use the attribute list as an alternate way to specify a volume label. That is, you can omit the volume label in the DEVICE\_SPEC argument and specify it in the ATTRIBUTE\_LIST. However, if you specify the volume label in both arguments, PROFBI overrides the ATTRIBUTE\_LIST specification with the label specified in DEVICE\_SPEC.

6. The attribute list for PROFBI also contains two additional, contiguous words as the Allocate Checkpoint Space (ACS) buffer. The high byte in the first word of the ACS buffer (2) identifies it as the ACS buffer. The low byte in the buffer specifies the number of bytes in that buffer. The second word in the ACS block identifies the number of blocks in the checkpoint file.

#### 8.4.2 Status Codes Returned by PROFBI

The status codes returned by PROFBI are listed in Table 8-2:

Table 8-2  
PROFBI Status Codes

<i>Status code</i>	<i>Comment</i>
+1	SUCCESS
-1	ILLEGAL DEVICE
-2	DEVICE NOT IN SYSTEM
-3	FAILED TO ATTACH DEVICE
-4	BLOCK ZERO BAD—DISK UNUSABLE
-5	AT LEAST ONE LBN (0 THROUGH 25) IS BAD CANNOT INITIALIZE—DISK UNUSABLE
-6	BAD BLOCK FILE OVERFLOW

## 8-10 CALLABLE SYSTEM ROUTINES

Table 8-2 (Cont.)

<i>Status Code</i>	<i>Comment</i>
-7	UNRECOVERABLE ERROR
-8.	DEVICE WRITE-LOCKED
-9.	DEVICE NOT READY
-10.	FAILED TO WRITE BAD BLOCK FILE
-11.	PRIVILEGE VIOLATION
-12.	DEVICE IS AN ALIGNMENT CARTRIDGE
-13.	FATAL HARDWARE ERROR
-14.	ALLOCATION FAILURE
-15.	I/O ERROR SIZING DEVICE
-16.	ALLOCATION FOR SYS FILE EXCEEDS VOLUME LIMIT
-17.	HOMEBLOCK ALLOCATE WRITE ERROR
-18.	BOOTBLOCK WRITE ERROR—DISK UNUSABLE
-19.	INDEX FILE BITMAP I/O ERROR
-20.	BAD BLOCK HEADER I/O ERROR
-21.	MFD FILE HEADER I/O ERROR
-22.	NULL FILE HEADER I/O ERROR
-23.	CHECKPOINT FILE HEADER I/O ERROR
-24.	MFD WRITE ERROR
-25.	STORAGE BITMAP FILE HEADER I/O ERROR
-26.	FAILED TO READ BAD BLOCK DESCRIPTOR FILE
-27.	VOLUME NAME TOO LONG
-28.	UNRECOGNIZED DISK TYPE
-29.	PREALLOCATION INSUFFICIENT TO FILL FIRST INDEX FILE HEADER
-30.	PREALLOCATED TOO MANY HEADERS FOR SINGLE HEADER INDEX FILE
-31.	PREALLOCATION INSUFFICIENT TO FILL FIRST AND SECOND INDEX FILE HEADERS
-32.	BAD BLOCK LIMIT EXCEEDED FOR DEVICE
-33.	DRIVER NOT RESIDENT
-34.	BITMAP TOO LARGE—INCREASE CLUSTER FACTOR
-35.	STORAGE BITMAP I/O ERROR
-36.	HOMEBLOCK I/O ERROR
-37.	INDEX FILE HEADER I/O ERROR
-38.	DISMOUNT OF DEVICE FAILED
-39.	CANNOT MOUNT DEVICE FOREIGN
-40.	CANNOT MOUNT DEVICE FILES-11
-41.	CANNOT FORMAT DZ—PREFORMATTED
-42.	CANNOT DETACH DEVICE
-43.	CHECKPOINT FILE HEADER OVERFLOW—SPECIFY SMALLER CHECKPOINT FILE
-44.	NON-ALPHANUMERIC CHARACTER(S) IN VOLUME NAME—ILLEGAL

## 8.5 PROLOG

The PROLOG routine provides five forms of logical name manipulation. You can use PROLOG as follows:

- Create a logical name for a device specification
- Delete a logical name for a device specification
- Translate a logical name to a device specification
- Set the default directory and/or device
- Show the default directory and device

The name of the server used to execute PROLOG is SUMLOG. This server must be installed in your system to perform any of the indicated services. Otherwise, PROLOG returns a directive error in the Status Control Block (see Section 8.1.3).

Caution: Do not use logical or directory names with this routine that are used by the P/OS system.

### 8.5.1 Creating or Translating a Logical Name

To create or translate a logical name, invoke the PROLOG routine with the following arguments:

```
STATUS, REQUEST, LOGICAL_NAME, LOGICAL_NAME_SIZE,
EQUIVALENCE, EQUIVALENCE_SIZE
```

where:

STATUS	The address of the 8-word Status Control Block				
REQUEST	The address of a word containing the decimal value indicating the operation (CREATE or TRANSLATE) to perform (see Section 8.3.3)				
LOGICAL_NAME	The address of a buffer containing an ASCII string (which can contain alphanumeric characters only)				
LOGICAL_NAME_SIZE	The address of a byte value containing the length of the string in LOGICAL_NAME				
EQUIVALENCE	The address of a buffer containing an ASCII device specification The device specification takes the form ddn: where: <table style="margin-left: 40px;"> <tr> <td>dd</td> <td>the device name</td> </tr> <tr> <td>n</td> <td>the device unit number</td> </tr> </table>	dd	the device name	n	the device unit number
dd	the device name				
n	the device unit number				
EQUIVALENCE_SIZE	For CREATE: The address of a byte value containing the length of the string in EQUIVALENCE. For TRANSLATE: The address for a byte value containing the length of the EQUIVALENCE buffer.				

For the TRANSLATE function, the EQUIVALENCE argument is an output argument returned by PROLOG. The length of the string returned in the EQUIVALENCE buffer is returned in the third word of STATUS.

### 8.5.2 Deleting a Logical Name and Set/Show

To delete a logical name or to set or show the default device and/or directory, invoke the PROLOG routine with the following arguments:

STATUS, REQUEST, LOGICAL\_NAME, LOGICAL\_NAME\_SIZE

where:

STATUS	The address of the 8-word Status Control Block
REQUEST	The address of a word containing the decimal value indicating the operation (DELETE, SET OR SHOW) to perform (see Section 8.3.3)
LOGICAL_NAME	The address of a buffer containing an ASCII string (which can contain alphanumeric characters only). The user must have already created the LOGICAL_NAME.
LOGICAL_NAME_SIZE	For SET and DELETE: The address of a byte value containing the length of the string in LOGICAL_NAME. For SHOW: The address of a byte value containing the length of the LOGICAL_NAME buffer.

For the SET DEFAULT function, the LOGICAL\_NAME string may contain a directory specification of the form

USERDISK:[DIRECTORY]

where USERDISK: is the logical name with the directory specification appended to it.

The directory specification takes one of the following forms:

[ggg,mmm]	such as [301,3]
	or
[gggmmm]	such as [301003]
	or
[name]	such as [WILEY]

where:

ggg	group
mmm	number

Note: When issuing a call for the SET DEFAULT function, note that the user can specify either the logical name or the directory. If you specify both, then both the default device and directory are changed. If you only specify one, the other does not change.



For both the DELETE and SET DEFAULT functions, there is no output argument; PROLOG returns the call status in the Status Control Block.

For the SHOW DEFAULT function, LOGICAL\_NAME is an output argument returned by PROLOG. The LOGICAL\_NAME also contains the default directory string. The length of the string returned in LOGICAL\_NAME is returned in the third word of STATUS.

The following is a sample BASIC-PLUS-2 call to PROLOG:

```
100 CALL PROLOG BY REF
   (STATUS%(),REQUEST%,DLOG$,LEN(DLOG$),EQV$,LEN(EQV$))
```

### 8.5.3 Using the REQUEST Argument for Create, Translate, Delete, Set or Show

PROLOG uses the value specified in the REQUEST argument to determine whether to create, translate, delete, set or show as follows:

- 1 SET DEFAULT
- 2 SHO DEFAULT
- 3 CREATE logical
- 4 TRANSLATE logical
- 5 DELETE logical

Most error returns from PROLOG are Directive Status errors (see CLOG\$ and DLOG\$ logical name directives in Chapter 9).

Table 8-3

<i>Status Code</i>	<i>Comment</i>
+1	SUCCESSFUL INSTALL
-1	TASK NAME IN USE
-3	SPECIFIED PARTITION TOO SMALL
-4	TASK AND PARTITION BASE MISMATCH
-7	LENGTH MISMATCH COMMON BLOCK
-8.	BASE MISMATCH COMMON BLOCK
-9.	TOO MANY COMMON BLOCK REQUESTS
-11.	CHECKPOINT AREA TOO SMALL
-13.	NOT ENOUGH APRS FOR TASK IMAGE
-14.	FILE NOT A TASK IMAGE
-15.	BASE ADDRESS MUST BE ON 4K BOUNDARY
-16.	ILLEGAL FIRST APR
-18.	COMMON BLOCK PARAMETER MISMATCH

Table 8-3 (Cont.)

<i>Status Code</i>	<i>Comment</i>
-20.	COMMON BLOCK NOT LOADED
-22.	TASK IMAGE VIRTUAL ADDRESS OVERLAPS COMMON BLOCK
-23.	TASK IMAGE ALREADY INSTALLED
-24.	ADDRESS EXTENSIONS NOT SUPPORTED
-26.	CHECKPOINT SPACE TOO SMALL, USING CHECKPOINT FILE
-27.	NO CHECKPOINT SPACE, ASSUMING NOT CHECKPOINTABLE
-29.	ILLEGAL UIC
-30.	NO POOL SPACE
-31.	ILLEGAL USE OF PARTITION OR REGION
-32.	ACCESS TO COMMON BLOCK DENIED
-33.	TASK IMAGE I/O ERROR
-34.	TOO MANY LUNS
-35.	ILLEGAL DEVICE
-36.	TASK MAY NOT BE RUN
-37.	TASK ACTIVE
-39.	TASK FIXED
-40.	TASK BEING FIXED
-41.	PARTITION BUSY
-43.	COMMON/TASK NOT IN SYSTEM
-44.	REGION OR COMMON FIXED
-45.	CANNOT DO RECEIVE
-47.	INVALID REQUEST
-48.	CANNOT RETURN STATUS
-49.	ERROR ENCOUNTERED ON FILE OPEN OPERATION
-50.	ERROR ENCOUNTERED ON FILE CLOSE OPERATION
-51.	CANNOT GET FILE LBN TO PROCESS LABEL BLOCKS

## 8.6 PROVOL

The PROVOL routine provides a twofold service. You can use the PROVOL routine to mount or dismount disk volumes. You can also use PROVOL to write a bootblock on a volume and/or bootstrap a volume.

To mount or dismount a volume, write a bootblock on a volume, or bootstrap a volume, invoke the PROVOL routine with the following arguments:

STATUS, REQUEST, DEVICE\_SPEC, DEVICE\_SIZE,  
ATTRIBUTE\_LIST, ATTRIBUTE\_SIZE

where:

STATUS	The address of the 8-word Status Control Block. When mounting or dismounting a volume, the last six words of the Status Control Block contain the volume label (provided that the operation is successful)
REQUEST	The address of a word containing the decimal value indicating the operation to be performed (see Section 8.6.1)
DEVICE_SPEC	The address of a buffer containing a character string which is the device specification of the volume to be mounted, dismounted, bootstrapped, or on which a bootblock is to be written
DEVICE_SIZE	The address of a word containing the length of the string in DEVICE_SPEC
ATTRIBUTE_LIST	The address of the attribute list. The attribute list is a buffer of legal attributes, predominantly intended for use by Macro programmers (see Notes). Legal attributes in PROVOL are: <ul style="list-style-type: none"> <li>1 Volume label</li> </ul>
ATTRIBUTE_SIZE	The address of a word containing the size of the attribute list

Note: The contents of the buffer for the ATTRIBUTE\_LIST argument are optional. That is, you must specify the argument but the buffer need not contain a volume label.

### 8.6.1 Using the REQUEST Argument in PROVOL

The PROVOL routine uses the decimal value specified in the REQUEST argument to determine which operation to perform. Specify in the REQUEST argument the value listed below that corresponds to the operation you desire:

0	Mount a volume
1	Mount a foreign volume
2	Dismount a volume
10	Bootstrap a volume
11	Write a bootblock on a volume
12	Write a bootblock on a volume and bootstrap it

#### Notes

1. The minimum length of DEVICE\_SPEC is four characters—the-three-character device mnemonic followed by a colon (such as DW1:). The device portion of DEVICE\_SPEC must end with a colon.

Part of the device specification can be the volume label which may be up to 12 characters. If you omit the volume label from `DEVICE_SPEC`, `PROVOL` gets the label from the specified disk by default. Whenever you specify a volume label in a `DEVICE_SPEC` argument (when mounting a volume, for example), the specified label must match the label on the volume; otherwise, the operation fails.

2. `DEVICE_SPEC` may also be a logical name string. In this case, the logical name string must end with a colon. The number of logical name translations cannot exceed eight. A ninth translation results in an error condition.
3. `PROVOL` requires the string supplied in the `DEVICE_SPEC` and `DEVICE_SIZE` arguments when mounting or dismounting a volume. The specified volume label must match the label on the volume for the operation to be successful. `PROVOL` ignores the volume label if mounting or dismounting a "foreign" volume.
4. `PROVOL` stores the volume label in the last six words of the Status Control Block when a mount or dismount is successful.
5. `PROVOL` uses the string supplied in the `DEVICE_SPEC` and `DEVICE_SIZE` arguments to bootstrap a volume. The `DEVICE_SPEC` string may be a logical name.
6. When writing a bootblock to a volume, `PROVOL` requires a complete device, directory and file name specification. If you omit the file name, `PROVOL` uses the default directory and file name of `[1,54]RSX11M.SYS`.
7. The `ATTRIBUTE_LIST` argument is the means of specifying optional parameters. The attribute list for `PROVOL` is simply a buffer of legal attributes. The high byte in the first word of the attribute list specifies the attribute type. The low byte specifies the size of the buffer in bytes.

You can use the attribute list as an alternate way to specify a volume label. That is, you can omit the volume label in the `DEVICE_SPEC` argument and supply it in the `ATTRIBUTE_LIST` argument. However, if you specify the volume label in both arguments, `PROVOL` overrides the `ATTRIBUTE_LIST` specification with the label specified in `DEVICE_SPEC`.

## CHAPTER 9

# DIRECTIVE DESCRIPTIONS

---

This chapter defines each of the system directives. The chapter describes each directive's function and use. For each directive there is also a description of the names of the corresponding macro and Fortran calls, the associated parameters, and possible return values of the Directive Status Word (DSW).

The descriptions generally show the \$ form of the macro call, although the \$C and \$\$ forms are also valid forms of the directive macro. (The QIO directive documents the QIO\$ form, although the QIO\$\$ and QIO\$C forms are also valid.) Where the \$\$ form of a macro requires less space and performs as fast as a DIR\$ (because of a small Directive Parameter Block), the documentation shows the \$\$ form of the macro expansion.

In addition to the directive macros themselves, you can use the DIR\$ macro to execute a directive if the directive has a predefined Directive Parameter Block (DPB). See Sections 3.3.1.1 and 3.3.2 for further details.

### 9.1 FORMAT OF SYSTEM DIRECTIVE DESCRIPTIONS

Each directive description includes most or all of the following elements:

#### **Fortran Call**

This shows the Fortran subroutine call, and defines each parameter. Programs written in other higher-level languages which provide support for the PDP-11 standard R5 calling conventions for Fortran may also make use of these calls. Check your language reference manual and user's guide to determine if you are using that format.

### Macro Call

This shows the macro call, defines each parameter, and gives the defaults for optional parameters in parentheses following the definition of the parameter. Since zero is supplied for most defaulted parameters, only nonzero default values are shown.

### Macro Expansion

Most of the directive descriptions expand the \$e form of the macro. Where the \$S form is the recommended form for a directive, the documentation shows that form of the macro expansion instead. Section 3.3.5 illustrates expansions for all three forms and for the DIR\$ macro.

### Definition Block Parameters

Only the memory management directive descriptions include these parameters. This section describes all the relevant input and output parameters in the Region or Window Definition Block (see Section 7.5).

### Local Symbol Definitions

Macro expansions usually generate local symbol definitions with an assigned value equal to the byte offset from the start of the DPB to the corresponding DPB element. This section lists those symbols. The length in bytes of the element pointed to by the symbol appears in parentheses following the symbol's description. Thus:

A.BTTN        task name (4)

defines A.BTTN as pointing to a task name in the Abort Task DPB; the task name has a length of four bytes.

### DSW Return Codes

This section lists all valid return codes.

### Notes

The notes presented with some directive descriptions expand on the function, use, and/or consequences of using the directives. Always read the notes carefully.

**ABRT\$****9.1.1 ABRT\$—Abort Task**

The Abort Task directive instructs the system to terminate the execution of the indicated task. ABRT\$ is intended for use as an emergency or fault exit.

A task must be privileged to issue the Abort Task directive (unless it is aborting a task with the same TI:).

**Fortran Call**

```
CALL ABORT (tsk[,ids])
```

tsk name of the task to be aborted (RAD50)

ids directive status

**Macro Call**

```
ABRT$ tsk
```

tsk name of the task to be aborted (RAD50)

**Macro Expansion**

```
ABRT$ ALPHA
.BYTE 83.,3 ;ABRT$ MACRO DIC, DPB SIZE=3 WORDS
.RAD50 /ALPHA/ ;TASK ''ALPHA''
```

**Local Symbol Definitions**

A.BTTN task name (4)

**DSW Return Codes**

IS.SUC successful completion

IE.INS task not installed

IE.ACT task not active

IE.PRI issuing task is not privileged (multiuser protection systems only)

IE.ADP part of the DPB is out of the issuing task's address space

IE.SDP directive Identification Code (DIC) or DPB size is invalid

**Notes**

1. When a task is aborted, the Executive frees all the task's resources. In particular, the Executive:
  - Detaches all attached devices.
  - Flushes the AST queue and despecifies all specified ASTs.
  - Flushes the receive and receive-by-reference queue.
  - Flushes the clock queue for outstanding Mark Time requests for the task.
  - Closes all open files (files open for write access are locked).
  - Detaches all attached regions except in the case of a fixed task, where no detaching occurs.
  - Runs down the task's I/O.
  - Disconnects from interrupt vectors.
  - Breaks the connection with any offspring tasks.
  - Returns a severe error status (EX\$SEV) to the parent task when a connected task is aborted.
  - Frees the task's memory if the aborted task was not fixed.
2. If the aborted task had a requested exit AST specified, the task will receive that AST instead of being aborted. No indication that this has occurred is returned to the task that issued the abort request.
3. When the aborted task actually exits, the Executive declares a significant event.



**ALTP\$****9.1.2 ALTP\$—Alter Priority**

The Alter Priority directive instructs the system to change the running priority of a specified active task to either a new priority indicated in the directive call, or to the task's default (installed) priority if the call does not specify a new priority.

The specified task must be installed and active. The Executive resets the task's priority to its installed priority when the task exits.

If the directive call omits a task name, the Executive defaults to the issuing task.

The Executive reorders any outstanding I/O requests for the task in the I/O queue and reallocates the task's partition. The partition reallocation may cause the task to be checkpointed.

A nonprivileged task can issue ALTP\$ only for itself, and only for a priority equal to or lower than its installed priority. A privileged task can change the priority of any task to any value less than 250.

**Fortran Call**

```
CALL ALTPRI ([tsk],[ipri][,ids])
```

tsk     active task name  
ipri    a 1-word integer value equal to the new priority, a number from 1 through 250(10)  
ids     directive status

**Macro Call**

```
ALTP$ [tsk][,pri]
```

tsk     active task name  
pri     new priority, a number from 1 through 250(10)

**Macro Expansion**

```
ALTP$     ALPHA, 75.  
.BYTE     9.,4         ;ALTP$ MACRO DIC, DPB SIZE=4 WORDS  
.RAD50    /ALPHA/     ;TASK ALPHA  
.WORD     75.         ;NEW PRIORITY
```

**Local Symbol Definitions**

A.LTTN        task name (4)

A.LTPR        priority (2)

**DSW Return Codes**

IS.SUC        successful completion

IE.INS        task not installed

IE.ACT        task not active

IE.PRI        issuing task is not privileged

IE.IPR        invalid priority

IE.ADP        part of the DPB is out of the issuing task's address space

IE.SDP        DIC or DPB size is invalid

**ALUN\$****9.1.3 ALUN\$—Assign LUN**

The Assign LUN directive instructs the system to assign a physical device unit to a logical unit number (LUN). It does not indicate that the task has attached itself to the device.

The actual physical device assigned to the logical unit is dependent on the logical assignment table. The Executive first searches the logical assignment table for a device name match. If it finds a match, the Executive assigns the physical device unit associated with the matching entry to the logical unit. Otherwise, the Executive searches the physical device tables and assigns the actual physical device unit named to the logical unit. The Executive does not search the logical assignment table for slaved tasks.

When a task reassigns a LUN from one device to another, the Executive cancels all I/O requests for the issuing task in the previous device queue.

**Fortran Call**

```
CALL ASNLUN (lun,dev,unt[,ids])
```

lun	logical unit number
dev	device name (format: 1A2)
unt	device unit number
ids	directive status

**Macro Call**

```
ALUN$ lun,dev,unt
```

lun	logical unit number
dev	device name (two characters)
unt	device unit number

**Macro Expansion**

ALUN\$	7,TT,0	;ASSIGN LOGICAL UNIT NUMBER
.BYTE	7,4	;ALUN\$ MACRO DIC, DPB SIZE=4 WORDS
.WORD	7	;LOGICAL UNIT NUMBER 7
.ASCII	/TT/	;DEVICE NAME IS TT (TERMINAL)
.WORD	0	;DEVICE UNIT NUMBER=0

### Local Symbol Definitions

A.LULU	logical unit number (2)
A.LUNA	physical device name (2)
A.LUNU	physical device unit number (2)

### DSW Return Codes

IS.SUC	successful completion
IE.LNL	LUN usage is interlocked (see Note 1 below)
IE.IDU	invalid device and/or unit
IE.ILU	invalid logical unit number
IE.ADP	part of the DPB is out of the issuing task's address space
IE.SDP	DIC or DPB size is invalid

### Notes

1. A return code of IE.LNL indicates that the specified LUN cannot be assigned as directed. Either the LUN is already assigned to a device with a file open for that LUN, or the LUN is currently assigned to a device attached to the task, and the directive attempted to change the LUN assignment. If a task has a LUN assigned to a device and the task has attached the device, the LUN can be reassigned, provided that the task has another LUN assigned to the same device.

## ASTX\$\$

### 9.1.4 ASTX\$\$—AST Service Exit (\$S Form Recommended)

The AST Service Exit directive instructs the system to terminate execution of an AST service routine.

If another AST is queued and ASTs are not disabled, then the Executive immediately effects the next AST. Otherwise, the Executive restores the task's pre-AST state. See Notes.

#### Fortran Call

Neither the Fortran language nor the ISA standard permits direct linking to system trapping mechanisms. (Refer to Section 3.4.4 for more information on this subject). Therefore, this directive is not available to Fortran tasks.

#### Macro Call

```
ASTX$$ [err]
```

err error routine address

#### Macro Expansion

```
ASTX$$ ERR
MOV      (PC)+, -(SP)      ;PUSH DPB ONTO THE STACK
.BYTE   115., 1           ;ASTX$$ MACRO DIC, DPB SIZE=1 WORD
EMT      377              ;TRAP TO THE EXECUTIVE
JSR      PC,ERR           ;CALL ROUTINE ''ERR'' IF DIRECTIVE
                          ;UNSUCCESSFUL
```

#### Local Symbol Definitions

None

#### DSW Return Codes

IS.SUC	successful completion
IE.AST	directive not issued from an AST service routine
IE.ADP	part of the DPB or stack is out of the issuing task's address space
IE.SDP	DIC or DPB size is invalid

## Notes

1. A return to the AST service routine occurs if, and only if, the directive is rejected. Therefore, no Branch On Carry Clear instruction is generated if an error routine address is given. (The return occurs only when the Carry bit is set.)
2. When an AST occurs, the Executive pushes, at minimum, the following information onto the task's stack:

- SP+06 event flag mask word
- SP+04 PS of task prior to AST
- SP+02 PC of task prior to AST
- SP+00 DSW of task prior to AST

The task stack must be in this state when the AST Service Exit directive is executed.

In addition to the data parameters, the Executive pushes supplemental information onto the task stack for certain ASTs. For I/O completion, the stack contains the address of the I/O status block; for Mark Time, the stack contains the Event Flag Number; for a floating-point processor exception, the stack contains the exception code and address.

These AST parameters must be removed from the task's stack prior to issuing an AST exit directive. The following example shows how to remove AST parameters when a task uses an AST routine on I/O completion:

```

;
; EXAMPLE PROGRAM
;
; LOCAL DATA
;
IOSB:  .BLKW  2           ;I/O STATUS DOUBLEWORD
BUFFER: .BLKW  30.       ;I/O BUFFER
;
; START OF MAIN PROGRAM
;
START:  .                ;PROCESS DATA
        .
        .
        QIOW$C  IO.WVB,2,1,,IOSB,ASTSER,<BUFFER,60.,40>
        .
        .                ;PROCESS & WAIT
        .
        .                ;EXIT TO EXECUTIVE
        EXIT$$
;
; AST SERVICE ROUTINE
;
ASTSER: .                ;PROCESS AST
        .
        .
        .
        TST     (SP)+    ;REMOVE ADDRESS OF I/O STATUS BLOCK
        ASTX$$ ;AST EXIT

```

3. The task can alter its return address by manipulating the information on its stack prior to executing an AST exit directive. For example, to return to task state at an address other than the pre-AST address indicated on the stack, the task can simply replace the PC word on the stack. This procedure may be useful in those cases in which error conditions are discovered in the AST routine; but you should use extreme caution when doing this alteration since AST service routine bugs are difficult to isolate.
4. Because this directive requires only a 1-word DPB, the \$\$ form of the macro is recommended. It requires less space and executes with the same speed as the DIR\$ macro.

## ATRG\$

### 9.1.5 ATRG\$—Attach Region

The Attach Region directive attaches the issuing task to a static common region or to a named dynamic region. (No other type of region can be attached to the task by means of this directive.) The Executive checks the desired access specified in the region status word against the owner UIC and the protection word of the region. If there is no protection violation, the Executive grants the desired access. If the region is successfully attached to the task, the Executive returns a 16-bit region ID (in R.GID), which the task uses in subsequent mapping directives.

You can also use the directive to determine the ID of a region already attached to the task. In this case, the task specifies the name of the attached region in R.GNAM and clears all four bits described below in the region status word R.GSTS. When the Executive processes the directive, it checks that the named region is attached. If the region is attached to the issuing task, the Executive returns the region ID, as well as the region size, for the task's first attachment to the region. You may want to use the Attach Region directive in this way to determine the region ID of a common block attached to the task at task-build time.

#### Fortran Call

```
CALL ATRG (irdb[,ids])
```

irdb an 8-word integer array containing a Region Definition Block (see Section 7.5.1.2)

ids directive status

#### Macro Call

```
ATRG$ rdb .
```

rdb region Definition Block (RDB) address

#### Macro Expansion

```
ATRG$ RDBADR
.BYTE 57.,2 ;ATRG$ MACRO DIC, DPB SIZE=2 WORDS
.WORD RDBADR ;RDB ADDRESS
```



Table 9-1  
Region Definition Block Parameters

<i>Input Parameters</i>		
<i>Array Element</i>	<i>Offset</i>	<i>Description</i>
irdb(3)(4)	R.GNAM	Name of the region to be attached
irdb(7)	R.GSTS	Bit settings <sup>1</sup> in the region status word (specifying desired access to the region):
	<i>Bit</i>	<i>Definition</i>
	RS.RED	1 if read access is desired
	RS.WRT	1 if write access is desired RS.EXT 1 if extend access is desired
	RS.DEL	1 if delete access is desired
		Clear all four bits to request the region ID of the named region if it is already attached to the issuing task.
<i>Output Parameters</i>		
irdb(1)	R.GID	ID assigned to the region
irdb(2)	R.GSIZ	Size in 32-word blocks of the attached region

### Local Symbol Definition

A.TRBA region Definition Block address (2)

### DSW Return Codes

IS.SUC successful completion  
 IE.UPN an attachment descriptor cannot be allocated  
 IE.PRI privilege violation  
 IE.NVR invalid region ID  
 IE.PNS specified region name does not exist  
 IE.HWR region had parity error or load failure  
 IE.ADP part of the DPB or RDB is out of the issuing task's address space  
 IE.SDP DIC or DPB size is invalid

1. If you are a FORTRAN programmer, refer to Section 7.5.1 to determine the bit values represented by the symbolic names described.

## CLEF\$

### 9.1.6 CLEF\$—Clear Event Flag

The Clear Event Flag directive instructs the system to report an indicated event flag's polarity and then clear it.

#### Fortran Call

```
CALL CLREF (efn[,ids])
```

efn event flag number

ids directive status

#### Macro Call

```
CLEF$ efn
```

efn event flag number

#### Macro Expansion

```
CLEF$ 52.
.BYTE 31.,2 ;CLEF$ MACRO DIC, DPB SIZE=2 WORDS
.WORD 52. ;EVENT FLAG NUMBER 52.
```

#### Local Symbol Definitions

C.LEEF event flag number (2)

#### DSW Return Codes

IS.CLR successful completion; flag was already clear

IS.SET successful completion; flag was set

IE.IEF invalid event flag number (EFN<1 or EFN>64)

IE.ADP part of the DPB is out of the issuing task's address space

IE.SDP DIC or DPB size is invalid

**CLOG\$****9.1.7 CLOG\$—Create Logical Name String**

The Create Logical Name String directive establishes the relationship between a logical name string and an equivalence name string. The maximum length for each string is  $255_{10}$  characters. If you create a logical name string with the same name as an existing logical name string, the new definition supersedes the old one.

**Fortran Call**

```
CALL CRELOG (mod,itbnum,lns,lnssz,iens,ienssz,idsw)
```

mod	the modifier of the logical name within a table
itbnum	the logical name table number: user (LT.USR) = 2  reserved for future use: task (LT.TSK) group (LT.GRP) system (LT.SYS)
lns	character array containing the logical name string
lnssz	size (in bytes) of the logical name string
iens	character array containing the equivalence name string
ienssz	size (in bytes) of the equivalence name string
idsw	integer to receive the Directive Status Word

**Macro Call**

```
CLOG$ mod,tbnum,lns,lnssz,ens,enssz
```

mod	the modifier of the logical name within a table
tbnum	the logical name table number: user (LT.USR) = 2  reserved for future use: task (LT.TSK) group (LT.GRP) system (LT.SYS)
lns	character array containing the logical name string
lnssz	size (in bytes) of the logical name string
iens	character array containing the equivalence name string
ienssz	size (in bytes) of the equivalence name string

**Macro Expansion**

```

CLOG$  MOD, TBNUM, LNS, LNSSZ, ENS, ENSSZ
.BYTE  207., 7          ;CLOG$ MACRO DIC, DPB SIZE = 7 WORDS
.BYTE  0                ;SUBFUNCTION
.BYTE  MOD              ;LOGICAL NAME MODIFIER
.BYTE  TBNUM           ;LOGICAL NAME TABLE NUMBER
.BYTE  0                ;RESERVED FOR FUTURE USE
.WORD  LNS             ;ADDRESS OF LOGICAL NAME BUFFER
.WORD  LNSSZ          ;BYTE COUNT OF LOGICAL NAME STRING
.WORD  ENS             ;ADDRESS OF EQUIVALENCE NAME BUFFER
.WORD  ENSSZ          ;BYTE COUNT OF EQUIVALENCE NAME STRING

```

**Local Symbol Definitions**

```

C.LENS      address of Equivalence name string (2)
C.LESZ      byte count of equivalence name string (2)
C.LFUN      subfunction (1)
C.LLNS      address of logical name string (2)
C.LLSZ      byte count of logical name string (2)
C.LMOD      logical name modifier (1)
C.LTBL      logical table number (1)

```

**DSW Return Codes**

```

IS.SUC      successful completion of service
IS.SUP      successful completion of service; a new equivalence name
            string superseded a previously specified name string
IE.UPN      insufficient dynamic storage is available to create the logical
            name
IE.IBS      the length of the logical or equivalence string is invalid; each
            string length must be greater than 0 but not greater than 25510
            characters
IE.ITN      invalid table number specified
IE.ADP      part of the DPB or user buffer is out of the issuing task's
            address space, or the user does not have proper access to that
            region
IE.SDP      DIC or DPB size is invalid

```

**CMKT\$****9.1.8 CMKT\$—Cancel Mark Time Requests**

The Cancel Mark Time Requests directive instructs the system to cancel a specific Mark Time Request or all Mark Time requests that have been made by the issuing task.

**Fortran Call**

```
CALL CANMT ([efn][,ids])
```

efn event flag number

ids directive status

**Macro Call**

```
CMKT$ [efn,ast,err]
```

err error routine address

efn event flag number

ast mark time AST address

**Macro Expansion**

```
CMKT$ 52.,MRKAST,ERR ;NOTE: THERE ARE TWO IGNORED ARGUMENTS
.BYTE 27.,3 ;CMKT$ MACRO DIC, DPB SIZE=3 WORDS
.WORD 52. ;EVENT FLAG NUMBER 52.
.WORD MRKAST ;ADDRESS OF MARK TIME REQUEST AST ROUTINE
```

Note: The above example will cancel only the Mark Time requests that were specified with efn 52 or the AST address.MRKAST. If no ast or efn parameters are specified, all Mark Time requests issued by the task are canceled, and the DPB size will equal 1.

**Local Symbol Definitions**

C.MKEF event flag number (2)

C.MKAE Mark Time Request AST routine address (2)

**DSW Return Codes**

IS.SUC successful completion

IE.ADP part of the DPB is out of the issuing task's address space

IE.SDP DIC or DPB size is invalid

## Notes

1. If neither the efn nor ast parameters are specified, all Mark Time Requests issued by the task are canceled. In addition, the DPB size will be one word. (When either the efn and/or ast parameters are specified, the DPB size will be three words.)
2. If both efn and ast parameters are specified (and nonzero), only Mark Time Requests issued by the task specifying either that event flag or AST address are canceled.
3. If only one efn or ast parameter is specified (and nonzero), only Mark Time Requests issued by the task specifying the event flag or AST address are canceled.

**CNCT\$****9.1.9 CNCT\$—Connect**

The Connect directive synchronizes the task issuing the directive with the exit or emit status of another task (offspring) that is already active. Execution of this directive queues an Offspring Control Block (OCB) to the offspring task, and increments the issuing task's rundown count (contained in the issuing task's Task Control Block). The rundown count is maintained to indicate the combined total number of tasks presently connected as offspring tasks and the total number of virtual terminals the task has created. The exit AST routine is called when the offspring exits or emits status with the address of the associated exit status block on the stack.

**Fortran Call**

CALL CNCT (rname,[iefn],[iast],[iesb],[iparm],[ids])

rname	single-precision, floating-point variable containing the offspring task name in Radix-50 format
iefn	event flag to be set when the offspring task exits or emits status
iast	name of an AST routine to be called when the offspring task exits or emits status

Note: Refer to Section 3.4.4 for important guidelines on using Fortran AST service routines.

iesb	name of an 8-word status block to be written when the offspring task exits or emits status
	Word 0          offspring task exit status
	Word 1          system abort code
	Word 2-7        reserved

Note: The exit status block defaults to one word. To use the 8-word exit status block, you must specify the logical OR of the symbol SP.WX8 and the event flag number in the iefn parameter above.

iparm	name of a word to receive the status block address when an AST occurs
ids	integer to receive the Directive Status Word

**Macro Call**

CNCT\$ tname,[efn],[east],[esb]

tname	name (RAD50) of the offspring task to be connected
efn	the event flag to be cleared on issuance and set when the offspring task exits or emits status

east            address of an AST routine to be called when the offspring task exits or emits status

esb            address of an 8-word status block to be written when the offspring task exits or emit status

                 Word 0            offspring task exit status

                 Word 1            system abort code

                 Word 2-7        reserved

Note: The exit status block defaults to one word. To use the 8-word exit status block, you must specify the logical OR of the symbol SP.WX8 and the event flag number in the efn parameter above.

**Macro Expansion**

```

CNCT$        ALPHA, 1, CONAST, STBUF
.BYTE        143., 6                ;CNCT$ MACRO DIC, DPB SIZE=6 WORDS
.RAD50       ALPHA                ;OFFSPRING TASK NAME
.BYTE        1                    ;EVENT FLAG NO = 1
.BYTE        16.                  ;EXIT STATUS BLOCK CONSTANT
.WORD        CONAST               ;AST ROUTINE ADDRESS
.WORD        STBUF                ;EXIT STATUS BLOCK ADDRESS
    
```

**Local Symbol Definitions**

C.NCTN       task name (4)

C.NCEF       event flag (2)

C.NCEA       AST routine address (2)

C.NCES       exit status block address (2)

**DSW Return Codes**

IS.SUC       successful completion

IE.UPN       insufficient dynamic memory to allocate an offspring control block

IE.INS       the specified task was a command line interpreter

IE.ACT       the specified task was not active

IE.IEF       invalid event flag number (EFN<0 or EFN>64)

IE.ADP       part of the DPB or exit status block is not in the issuing task's address space

IE.SDP       DIC or DPB size is invalid

**Note**

1. Do not change the virtual mapping of the exit status block while the connection is in effect. Doing so may cause obscure errors since the exit status block is always returned to the virtual address specified regardless of the physical address to which it is mapped.



## CRAW\$

### 9.1.10 CRAW\$—Create Address Window

The `Create_Address_Window` directive creates a new virtual address window by allocating a window block from the header of the issuing task and establishing its virtual address base and size. (Space for the window block has to be reserved at task-build time by means of the `WNDWS` keyword. Execution of this directive unmaps and then eliminates any existing windows that overlap the specified range of virtual addresses. If the window is successfully created, the Executive returns an 8-bit window ID to the task.

The 8-bit window ID returned to the task is a number from 1 through 23, which is an index to the window block in the task's header. The window block describes the created address window.

If `WS.MAP` in the window status word is set, the Executive proceeds to map the window according to the Window Definition Block (WDB) input parameters.

A task can specify any length for the mapping assignment that is less than or equal to both the window size specified when the window was created, and the length remaining between the specified offset within the region and the end of the region.

If `W.NLEN` is set to 0, the length defaults to either the window size or the length remaining in the region, whichever is smaller. (Because the Executive returns the actual length mapped as an output parameter, the task must clear that offset before issuing the directive each time it wants to default the length of the map.)

The values that can be assigned to `W.NOFF` depend on the setting of bit `WS.64B` in the window status word (`W.NSTS`):

- If `WS.64B = 0`, the offset specified in `W.NOFF` must represent a multiple of 256 words (512 bytes). Because the value of `W.NOFF` is expressed in units of 32-word blocks, the value must be a multiple of 8.
- If `WS.64B = 1`, the task can align on 32-word boundaries; the programmer can therefore specify any offset within the region.

#### Fortran Call

```
CALL CRAW (iwdb[,ids])
```

`iwdb`    an 8-word integer array containing a Window Definition Block (see Section 7.5.2.2)

`ids`     directive status

**Macro Call**

CRAW\$ wdb

wdb Window Definition Block address

**Macro Expansion**

```
CRAW$   WDBADR
.BYTE   117.,2   ;CRAW$ MACRO DIC, DPB SIZE=2 WORDS
.WORD   WDBADR   ;WDB ADDRESS
```

Table 9-2  
Window Definition Block Parameters

*Input Parameters*

<i>Array Element</i>	<i>Offset</i>	<i>Description</i>								
iwdb(1), bits 8-15	W.NAPR	base APR of the address window to be created								
iwdb(3)	W.NSIZ	desired size, in 32-word blocks, of the address window								
iwdb(4)	W.NRID	ID of the region to which the new window is to be mapped, or 0 for task region (to be specified only if WS.MAP=1)								
iwdb(5)	W.NOFF	offset in 32-word blocks from the start of the region at which the window is to start mapping (to be specified only if WS.MAP=1). Note: If WS.64B in the window status word equals 0, the value specified must be a multiple of 8.								
iwdb(6)	W.NLEN	length in 32-word blocks to be mapped, or 0 if the length is to default to either the size of the window or the space remaining in the region, whichever is smaller (to be specified only if WS.MAP=1)								
iwdb(7)	W.NSTS	bit settings <sup>2</sup> in the window status word: <table border="1" data-bbox="483 1381 1130 1579"> <thead> <tr> <th><i>Bit</i></th> <th><i>Definition</i></th> </tr> </thead> <tbody> <tr> <td>WS.MAP</td> <td>1 if the new window is to be mapped</td> </tr> <tr> <td>WS.WRT</td> <td>1 if the mapping assignment is to occur with write access</td> </tr> <tr> <td>WS.64B</td> <td>0 for 256-word (512-byte) alignment; or 1 for 32-word (64-byte) alignment</td> </tr> </tbody> </table>	<i>Bit</i>	<i>Definition</i>	WS.MAP	1 if the new window is to be mapped	WS.WRT	1 if the mapping assignment is to occur with write access	WS.64B	0 for 256-word (512-byte) alignment; or 1 for 32-word (64-byte) alignment
<i>Bit</i>	<i>Definition</i>									
WS.MAP	1 if the new window is to be mapped									
WS.WRT	1 if the mapping assignment is to occur with write access									
WS.64B	0 for 256-word (512-byte) alignment; or 1 for 32-word (64-byte) alignment									

2. If you are a Fortran programmer, refer to Section 7.5.2 to determine the bit values represented by the symbolic names described.

Table 9-2 (Cont.)

*Output Parameters*

<i>Array Element</i>	<i>Offset</i>	<i>Description</i>
iwdb(1), bits 0-7	W.NID	ID assigned to the window
iwdb(2)	W.NBAS	virtual address base of the new window
iwdb(6)	W.NLEN	length, in 32-word blocks, actually mapped by the window
iwdb(7)	W.NSTS	bit settings <sup>2</sup> in the window status word:
	<i>Bit</i>	<i>Definition (if bit=1)</i>
	WS.CRW	address window was successfully created
	WS.UNM	at least one window was unmapped
	WS.ELW	at least one window was eliminated
	WS.RRF	reference was successfully received
	WS.RES	map only if resident
	WS.NAT	create attachment descriptor only if necessary (for Send By Reference directives)
	WS.64B	define the task's permitted alignment boundaries - 0 for 256-word (512-byte) alignment; or 1 for 32-word (64-byte) alignment
	WS.MAP	window is to be mapped
	WS.RCX	exit if no references to receive
	WS.DEL	send with delete access
	WS.EXT	send with extend access
	WS.WRT	send with write access or map with write access
	WS.RED	send with read access

2. If you are a Fortran programmer, refer to Section 7.5.2 to determine the bit values represented by the symbolic names described.

### Local Symbol Definitions

C.RABA        window Definition Block address (2)

### DSW Return Codes

IS.SUC        successful completion

E.PRI        requested access denied at mapping stage

IE.NVR        invalid region ID

IE.ALG        task specified either an invalid base APR and window size combination, or an invalid region offset and length combination in the mapping assignment; or WS.64B = 0 and the value of W.NOFF is not a multiple of 8

IE.WOV        no window blocks available in task's header

IE.ADP        part of the DPB or WDB is out of the issuing task's address space

IE.SDP        DIC or DPB size is invalid

**CRRG\$****9.1.11 CRRG\$—Create Region**

The Create Region directive creates a dynamic region in a system-controlled partition and optionally attaches it to the issuing task.

If RS.ATT is set in the region status word, the Executive attempts to attach the task to the newly created region. If no region name has been specified, the user's program must set RS.ATT (see the description of the Attach Region directive).

By default, the Executive marks a dynamically created region for deletion when the last task detaches from it. To override this default condition, set RS.NDL in the region status word as an input parameter. Be careful in considering to override the delete-on-last-detach option. An error within a program can cause the system to lock by leaving no free space in a system-controlled partition.

If the region is not given a name, the Executive ignores the state of RS.NDL. All unnamed regions are deleted when the last task detaches from them.

Named regions are put in the Common Block Directory (CBD). However, memory is not allocated until the Executive maps a task to the region.

The Executive returns an error if there is not enough space to accommodate the region in the specified partition. (See Notes.)

**Fortran Call**

```
CALL CRRG (irdb[,ids])
```

irdb an 8-word integer array containing a Region Definition Block (see Section 7.5.1.2)

ids directive status

**Macro Call**

```
CRRG$rdb
```

rdb Region Definition Block address

**Macro Expansion**

```
CRRG$  RDBADR
.BYTE  55.,2      ;CRRG$ MACRO DIC, DPB SIZE = 2 WORDS
.WORD  RDBADR     ;RDB ADDRESS
```

Table 9-3  
Region Definition Block Parameters

*Input Parameters*

<i>Array Element</i>	<i>Offset</i>	<i>Description</i>																						
irdb	R.GSIZ	size, in 32-word blocks, of the region to be created																						
irdb(3)(4)	R.GNAM	name of the region to be created, or 0 for no name																						
irdb(5)(6)	R.GPAR	name of the system-controlled partition in which the region is to be allocated, or 0 for the partition in which the task is running																						
irdb(7)	R.GSTS	bit settings <sup>3</sup> in the region status word:  <table border="1"> <thead> <tr> <th><i>Bit</i></th> <th><i>Definition (if bit=1)</i></th> </tr> </thead> <tbody> <tr> <td>RS.CRR</td> <td>region was successfully created</td> </tr> <tr> <td>RS.UNM</td> <td>at least one window was unmapped on a detach</td> </tr> <tr> <td>RS.MDL</td> <td>mark region for deletion on last detach</td> </tr> <tr> <td>RS.NDL</td> <td>the region should not be deleted on last detach</td> </tr> <tr> <td>RS.ATT</td> <td>created region should be attached</td> </tr> <tr> <td>RS.NEX</td> <td>created region is not extendible</td> </tr> <tr> <td>RS.RED</td> <td>read access is desired on attach</td> </tr> <tr> <td>RS.WRT</td> <td>write access is desired on attach</td> </tr> <tr> <td>RS.EXT</td> <td>extend access is desired on attach</td> </tr> <tr> <td>RS.DEL</td> <td>delete access is desired on attach</td> </tr> </tbody> </table>	<i>Bit</i>	<i>Definition (if bit=1)</i>	RS.CRR	region was successfully created	RS.UNM	at least one window was unmapped on a detach	RS.MDL	mark region for deletion on last detach	RS.NDL	the region should not be deleted on last detach	RS.ATT	created region should be attached	RS.NEX	created region is not extendible	RS.RED	read access is desired on attach	RS.WRT	write access is desired on attach	RS.EXT	extend access is desired on attach	RS.DEL	delete access is desired on attach
<i>Bit</i>	<i>Definition (if bit=1)</i>																							
RS.CRR	region was successfully created																							
RS.UNM	at least one window was unmapped on a detach																							
RS.MDL	mark region for deletion on last detach																							
RS.NDL	the region should not be deleted on last detach																							
RS.ATT	created region should be attached																							
RS.NEX	created region is not extendible																							
RS.RED	read access is desired on attach																							
RS.WRT	write access is desired on attach																							
RS.EXT	extend access is desired on attach																							
RS.DEL	delete access is desired on attach																							
irdb(8)	R.GPRO	protection word for the region (DEWR,DEWR,DEWR,DEWR)																						

*Output Parameters*

irdb(1)	R.GID	ID assigned to the created region (returned if RS.ATT=1)				
irdb(2)	R.GSIZ	size in 32-word blocks of the attached region (returned if RS.ATT=1)				
irdb(7)	R.GSTS	bit settings <sup>4</sup> in the region status word:  <table border="1"> <thead> <tr> <th><i>Bit</i></th> <th><i>Definition</i></th> </tr> </thead> <tbody> <tr> <td>RS.CRR</td> <td>1 if the region was successfully created</td> </tr> </tbody> </table>	<i>Bit</i>	<i>Definition</i>	RS.CRR	1 if the region was successfully created
<i>Bit</i>	<i>Definition</i>					
RS.CRR	1 if the region was successfully created					

3. If you are a Fortran programmer, refer to Section 7.5.1 to define the bit values represented by the symbolic names described.
4. If you are a FORTRAN programmer, refer to section 7.5.1 to determine the bit values represented by the symbolic names described.

**Local Symbol Definitions**

C.RRBA region Definition Block address (2)

**DSW Return Codes**

IS.SUC successful completion

IE.UPN a Partition Control Block (PCB) or an attachment descriptor could not be allocated, or the partition was not large enough to accommodate the region, or there is currently not enough continuous space in the partition to accommodate the region

IE.HWR the directive failed in the attachment stage because a region parity error was detected

IE.PRI attach failed because desired access was not allowed

IE.PNS specified partition in which the region was to be allocated does not exist; or no region name was specified and RS.ATT = 0

IE.ADP part of the DPB or RDB is out of issuing task's address space

IE.SDP DIC or RDB size is invalid

**Notes**

1. The Executive does not return an error if the named region already exists. In this case, the Executive clears the RS.CRR bit in the status word R.GSTS. If RS.ATT has been set, the Executive attempts to attach the already existing named region to the issuing task.
2. The protection word (see R.GPRO above) has the same format as that of the file system protection word. There are four categories, and the access for each category is coded into four bits. From low order to high order, the categories follow this order: system, owner, group, world. The access code bits within each category are arranged (from low order to high order) as follows: read, write, extend, delete. A bit that is set indicates that the corresponding access is denied.

The issuing task's UIC is the created region's owner UIC.

In order to prevent the creation of common blocks that are not easily deleted, the system and owner categories are always forced to have delete access, regardless of the value actually specified in the protection word.

## CSRQ\$

### 9.1.12 CSRQ\$—Cancel Time Based Initiation Requests

The Cancel Time Based Initiation Requests directive instructs the system to cancel all time-synchronized initiation requests for a specified task, regardless of the source of each request. These requests result from a Run directive.

#### Fortran Call

```
CALL CANALL (tsk[,ids])
```

tsk     task name  
ids     directive status

#### Macro Call

```
CSRQ$ tsk
```

tsk     scheduled (target) task name

#### Macro Expansion

```
CSRQ$   ALPHA  
.BYTE   25.,3               ;CSRQ$ MACRO DIC, DPB SIZE=3 WORDS  
.RAD50  /ALPHA/             ;TASK ''ALPHA''
```

#### Local Symbol Definitions

C.SRTN     target task name (4)

#### DSW Return Codes

IS.SUC     successful completion  
IE.INS     task is not installed  
IE.PRI     the issuing task is not privileged and is attempting to cancel requests made by another task  
IE.ADP     part of the DPB is out of the issuing task's address space  
IE.SDP     DIC or DPB size is invalid



**Note**

1. If you specify an error routine address when using the \$C or \$S macro form, you must include a null argument. For example:

```
CSRQ$S      #TNAME,,ERR      ;CANCEL REQUESTS FOR ''ALPHA''  
.  
.  
.  
TNAME: .RAD50 /ALPHA/
```

## DECL\$\$

### 9.1.13 DECL\$\$—Declare Significant Event (\$\$ Form Recommended)

The Declare Significant Event directive instructs the system to declare a significant event.

Declaration of a significant event causes the Executive to scan the Active Task List from the beginning, searching for the highest priority task that is ready to run. Use this directive with discretion to avoid excessive scanning overhead.

#### Fortran Call

```
CALL DECLAR ([,ids])
```

ids directive status

#### Macro Call

```
DECL$$ [,err]
```

err error routine address

#### Macro Expansion

```
DECL$$ ,ERR ;NOTE: THERE IS ONE IGNORED ARGUMENT
MOV (PC)+, -(SP) ;PUSH DPB ONTO THE STACK
.BYTE 35., 1 ;DECL$$ MACRO DIC, DPB SIZE=1 WORD
EMT 377 ;TRAP TO THE EXECUTIVE
BCC .+6 ;BRANCH IF DIRECTIVE SUCCESSFUL
JSR PC,ERR ;OTHERWISE, CALL ROUTINE ''ERR''
```

#### Local Symbol Definitions

None

#### DSW Return Codes

IS.SUC successful completion  
 IE.ADP part of the DPB is out of the issuing task's address space  
 IE.SDP DIC or DPB size is invalid

#### Note

1. The \$\$ form of the macro is recommended because this directive requires only a 1-word DPB.

**DLOG\$****9.1.14 DLOG\$—Delete Logical Name**

The Delete Logical Name directive deletes a logical name from the logical name table and returns to the system the resources used by that logical name. You should delete logical names when they are no longer needed. If you do not specify the the logical name string buffer address, DLOG\$ deletes all of the logical names within the specified logical name table.

**Fortran Call**

CALL DELLOG (mod,itbnum,lns,lnssz,idsw)

mod	the modifier of the logical name within a table
itbnum	the logical name table number: user (LT.USR) 2  reserved for future use: task (LT.TSK) group (LT.GRP) system (LT.SYS)
lns	character array containing the logical name string
lnssz	size (in bytes) of the logical name string
idsw	integer to receive the Directive Status Word

**Macro Call**

DLOG\$ mod,tbnum,lns,lnssz

mod	the modifier of the logical name within a table
tbnum	the logical name table number: user (LT.USR) 2  Reserved for future use: task (LT.TSK) group (LT.GRP) system (LT.SYS)
lns	character array containing the logical name string
lnssz	size (in bytes) of the logical name string

**Macro Expansion**

```

DLOG$  mod,tbnum,lns,lnssz
.BYTE  207.,5           ;DLOG$ MACRO DIC, DPB SIZE = 5 WORDS
.BYTE  2                ;SUBFUNCTION CODE FOR DELETION
.BYTE  MOD              ;LOGICAL NAME MODIFIER
.BYTE  TBNUM            ;LOGICAL NAME TABLE NUMBER
.BYTE  0                ;RESERVED FOR FUTURE USE
.WORD  LNS              ;ADDRESS OF THE LOGICAL NAME BUFFER
.WORD  LNSSZ            ;BYTE COUNT OF THE LOGICAL NAME STRING

```

**Local Symbol Definitions**

D.LFUN        subfunction (1)

D.LLNS        address of logical name string (2)

D.LLSZ        byte count of logical name string (2)

D.LMOD        logical name modifier (1)

D.LTBL        logical table number (1)

**DSW Return Codes**

IS.SUC        successful completion

IE.LNF        the specified logical name string was not found

IE.IBS        the length of the logical or equivalence string is invalid. Each string length must be greater than 0 but not greater than 255<sub>10</sub> characters

IE.ITN        invalid table number specified

IE.ADP        part of the DPB or user buffer is out of the issuing task's address space, or the user does not have proper access to that region

IE.SDP        DIC or DPB size is invalid

**Notes**

1. This directive disables only the recognition of ASTs; the Executive still queues the ASTs. They are queued FIFO and will occur in that order when the task reenables AST recognition.
2. Because this directive requires only a 1-word DPB, the \$S form of the macro is recommended. It requires less space and executes with the same speed as that of the DIR\$ macro.

**DSAR\$\$/IHAR\$\$****9.1.15 DSAR\$\$ or IHAR\$\$—Disable (or Inhibit) AST Recognition (\$S Form Recommended)**

The Disable (or Inhibit) AST Recognition directive instructs the system to disable recognition of ASTs for the issuing task. The ASTs are queued as they occur and are effected when the task reenables AST recognition. There is an implied disable AST recognition directive whenever an AST service routine is executing. When a task's execution is started, AST recognition is enabled. (See Notes.)

**Fortran Call**

```
CALL DSASTR [(ids)]
or
CALL INASTR [(ids)]
```

ids directive status

**Macro Call**

```
DSAR$$ [err]
```

err error routine address

**Macro Expansion:**

```
DSAR$$ ERR
MOV      (PC)+, -(SP)      ;PUSH DPB ONTO THE STACK
.BYTE    99., 1           ;DSAR$$ MACRO DIC, DPB SIZE=1 WORD
EMT      377              ;TRAP TO THE EXECUTIVE
BCC      .+6              ;BRANCH IF DIRECTIVE SUCCESSFUL
JSR      PC,ERR           ;OTHERWISE, CALL ROUTINE ''ERR''
```

**Local Symbol Definitions**

None

**DSW Return Codes**

IS.SUC	successful completion
IE.ITS	AST recognition is already disabled
IE.ADP	part of the DPB is out of the issuing task's address space
IE.SDP	DIC or DPB size is invalid

### Notes

1. This directive disables only the recognition of ASTs; the Executive still queues the ASTs. They are queued FIFO and will occur in that order when the task reenables AST recognition.
2. Because this directive requires only a 1-word DPB, the \$S form of the macro is recommended. It requires less space and executes with the same speed as that of the DIR\$ macro.

**DSCP\$\$****9.1.16 DSCP\$\$—Disable Checkpointing (\$S Form Recommended)**

The Disable Checkpointing directive instructs the system to disable checkpointing for a task that has been installed as a checkpointable task. Only the affected task can issue this directive. A task cannot disable the ability of another task to be checkpointed.

**Fortran Call**

```
CALL DISCKP [(ids)]
```

ids     directive status

**Macro Call**

```
DSCP$$ [err]
```

err     error routine address

**Macro Expansion**

```
DSCP$$  ERR
MOV     (PC)+, -(SP)    ;PUSH DPB ONTO THE STACK
.BYTE   95., 1         ;DSCP$$ MACRO DIC, DPB SIZE=1 WORD
EMT     377            ;TRAP TO THE EXECUTIVE
BCC     .+6            ;BRANCH IF DIRECTIVE SUCCESSFUL
JSR     PC,ERR         ;OTHERWISE, CALL ROUTINE ''ERR''
```

**Local Symbol Definitions**

None

**DSW Return Codes**

IS.SUC	successful completion
IE.ITS	task checkpointing is already disabled
IE.CKP	issuing task is not checkpointable
IE.ADP	part of the DPB is out of the issuing task's address space
IE.SDP	DIC or DPB size is invalid

**Notes**

1. When a checkpointable task's execution is started, checkpointing is enabled (that is, the task can be checkpointed).
2. Because this directive requires only a 1-word DPB, the \$S form of the macro is recommended. It requires less space and executes with the same speed as that of the DIR\$ macro.

## DTRG\$

### 9.1.17 DTRG\$—Detach Region

The Detach Region directive detaches the issuing task from a specified, previously attached region. Any of the task's windows that are currently mapped to the region are automatically unmapped.

If RS.MDL is set in the region status word when the directive is issued, the task marks the region for deletion on the last detach. A task must be attached with delete access to mark a region for deletion.

#### Fortran Call

```
CALL DTRG (irdb[,ids])
```

irdb    an 8-word integer array containing a Region Definition Block (see Section 7.5.1.2)  
ids     directive status

#### Macro Call

```
DTRG$ rdb
```

rdb     Region Definition Block address

#### Macro Expansion

```
DTRG$    RDBADR  
.BYTE    59.,2     ;DTRG$ MACRO DIC, DPB SIZE=2 WORDS  
.WORD    RDBADR   ;RDB ADDRESS
```

#### Local Symbol Definitions

D.TRBA     region Definition Block address (2)

#### DSW Return Codes

IS.SUC     successful completion  
IE.PRI     the task, which is not attached with delete access, has attempted to mark the region for deletion on the last detach, or the task has outstanding I/O  
IE.NVR     the task specified an invalid region ID or attempted to detach region 0 (its own task region)  
IE.ADP     part of the DPD or RDB is out of the issuing task's address space  
IE.SDP     DIC or DPB size is invalid



Table 9-4  
Region Definition Block Parameters

---

*Input Parameters*

---

<i>Array Element</i>	<i>Offset</i>	<i>Description</i>
irdb(1)	R.GID	ID of the region to be detached
irdb(7)	R.GSTS	bit settings <sup>5</sup> in the region status word:
		<i>Bit</i> <i>Definition</i>
		RS.MDL                  1 if the region should be marked for deletion when the last task detaches from it

---

*Output Parameters*

---

irdb(7)	R.GSTS	bit settings <sup>6</sup> in the region status word:
		<i>Bit</i> <i>Definition</i>
		RS.UNM                  1 if any windows were unmapped

---

5. If you are a Fortran programmer, refer to Section 7.5.1 to determine the bit values represented by the symbolic names described.
6. If you are a Fortran programmer, refer to Section 7.5.1 to determine the bit values represented by the symbolic names described.

## ELAW\$

### 9.1.18 ELAW\$—Eliminate Address Window

The Eliminate Address Window directive deletes an existing address window, unmapping it first if necessary. Subsequent use of the eliminated window's ID is invalid.

#### Fortran Call

```
CALL ELAW (iwdb[,ids])
```

**iwdb** an 8-word integer array containing a Window Definition Block (see Section 7.5.2.2)

**ids** directive status

#### Macro Call

```
ELAW$ wdb
```

**wdb** Window Definition Block address

#### Macro Expansion

```
ELAW$ WDBADR
.BYTE 119., 2 ;ELAW$ MACRO DIC, DPB SIZE=2 WORDS
.WORD WDBADR ;WDB ADDRESS
```

Table 9-5  
Window Definition Block Parameters

---

#### Input Parameters

---

##### Array

Element	Offset	Description
iwdb(1) bits 0-7	W.NID	ID of the address window to be eliminated

---

#### Output Parameters

---

iwdb(7)	W.NSTS	Bit settings <sup>7</sup> in the window status word:
	<i>Bit</i>	<i>Definition</i>
	WS.ELW	1 if the address window was successfully eliminated
	WS.UNM	1 if the address window was unmapped

---

7. If you are a Fortran programmer, refer to Section 7.5.2 to determine the bit values represented by the symbolic names described.

**Local Symbol Definitions**

E.LABA        Window Definition Block address (2)

**DSW Return Codes**

IS.SUC        successful completion  
IE.NVW        invalid address window ID  
IE.ADP        part of the DPB or WDB is out of the issuing task's address  
                 space  
IE.SDP        DIC or DPB size is invalid

## EMST\$

### 9.1.19 EMST\$—Emit Status

The Emit Status directive returns the specified 16-bit quantity to the specified connected task. It possibly sets an event flag or declares an AST if previously specified by the connected task in a Send, Request And Connect, a Spawn, or a Connect directive. If the specified task is multiply connected to the task issuing this directive, the first (oldest) Offspring Control Block (OCB) in the queue is used to return status. If no task name is specified, this action is taken for all tasks that are connected to the issuing task at that time. In any case, whenever status is emitted to one or more tasks, those tasks no longer remain connected to the task issuing the Emit Status directive.

#### Fortran Call

```
CALL EMST ([rtname],status[,ids])
```

rtname	name of a task connected to the issuing task to which the status is to be emitted
status	a 16-bit quantity to be returned to the connected task
ids	integer to receive the Directive Status Word

#### Macro Call

```
EMST$ [tname],status
```

tname	name of a task connected to the issuing task to which the status is to be emitted
status	16-bit quantity to be returned to the connected task

#### Macro Expansion

```
EMST$      ALPHA,STWD
.BYTE      147.,4      ;EMST$ MACRO DIC, DPB SIZE=4 WORDS
.RAD50     ALPHA      ;NAME OF CONNECTED TASK TO RECEIVE STATUS
.WORD      STWD       ;VALUE OF STATUS TO BE RETURNED
```

#### Local Symbol Definitions

E.MSTN	task name (4)
E.MSST	status to be returned (2)

**DSW Return Codes**

IS.SUC	successful completion
IE.ITS	the specified task is not connected to the issuing task
IE.ADP	part of the DPB is out of the issuing task's address space
IE.SDP	DIC or DPB size is invalid

## ENAR\$\$

### 9.1.20 ENAR\$\$—Enable AST Recognition (\$\$ Form Recommended)

The Enable AST Recognition directive instructs the system to recognize ASTs for the issuing task; that is, the directive nullifies a Disable AST Recognition directive. ASTs that were queued while recognition was disabled are effected at issuance. When a task's execution is started, AST recognition is enabled.

#### Fortran Call

```
CALL ENASTR [(ids)]
```

ids     directive status

#### Macro Call

```
ENAR$$ [err]
```

err     error routine address

#### Macro Expansion

```
ENAR$$ ERR
MOV      (PC)+, -(SP)      ;PUSH DPB ONTO THE STACK
.BYTE    101., 1           ;ENAR$$ MACRO DIC, DPB SIZE=1 WORD
EMT      377               ;TRAP TO THE EXECUTIVE
BCC      .+6               ;BRANCH IF DIRECTIVE SUCCESSFUL
JSR      PC,ERR            ;OTHERWISE, CALL ROUTINE ''ERR''
```

#### Local Symbol Definitions

None

#### DSW Return Codes

IS.SUC	successful completion
IE.ITS	AST recognition is not disabled
IE.ADP	part of the DPB is out of the issuing task's address space
IE.SDP	DIC or DPB size is invalid

#### Note

1. Because this directive requires only a 1-word DPB, the \$\$ form of the macro is recommended. It requires less space and executes with the same speed as that of the DIR\$ macro.

**ENCP\$\$****9.1.21 ENCP\$\$—Enable Checkpointing (\$S Form Recommended)**

The Enable Checkpointing directive instructs the system to make the issuing task checkpointable after its checkpointability has been disabled; that is, the directive nullifies a DSCP\$\$ directive. This directive cannot be used to enable checkpointing of a task that was built noncheckpointable.

**Fortran Call**

```
CALL ENACKP [(ids)]
```

ids     directive status

**Macro Call**

```
ENCP$$ [err]
```

err     error routine address

**Macro Expansion**

```
ENCP$$ ERR
MOV      (PC)+, -(SP)      ;PUSH DPB ONTO THE STACK
.BYTE   97., 1             ;ENCP$$ MACRO DIC, DPB SIZE=1 WORD
EMT     377                ;TRAP TO THE EXECUTIVE
BCC     .+6                ;BRANCH IF DIRECTIVE SUCCESSFUL
JSR     PC,ERR             ;OTHERWISE, CALL ROUTINE ''ERR''
```

**Local Symbol Definitions**

None

**DSW Return Codes**

IS.SUC	successful completion
IE.ITS	checkpointing is not disabled or task is connected to an interrupt vector
IE.ADP	part of the DPB is out of the issuing task's address space
IE.SDP	DIC or DPB size is invalid

**Note**

1. Because this directive requires only a 1-word DPB, the \$\$ form of the macro is recommended. It requires less space and executes with the same speed as that of the DIR\$ macro.

## EXIF\$

### 9.1.22 EXIF\$—Exit If

The Exit If directive instructs the system to terminate the execution of the issuing task if, and only if, an indicated event flag is not set. The Executive returns control to the issuing task if the specified event flag is set. See Notes.

#### Fortran Call

```
CALL EXITIF (efn[,ids])
```

efn    event flag number  
ids    directive status

#### Macro Call

```
EXIF$efn
```

efn    event flag number

#### Macro Expansion

```
EXIF$    52.  
.BYTE    53.,2                    ;EXIF$ MACRO DIC, DPB SIZE=2 WORDS  
.WORD    52.                     ;EVENT FLAG NUMBER 52.
```

#### Local Symbol Definitions

E.XFEF    event flag number (2)

#### DSW Return Codes

IS.SET    indicated EFN set; task did not exit  
IE.IEF    invalid event flag number (EFN<1 or EFN>64)  
IE.ADP    part of the DPB is out of the issuing task's address space  
IE.SDP    DIC or DPB size is invalid

#### Notes

1. The Exit If directive is useful in avoiding a possible race condition that can occur between two tasks communicating by means of the Send and Receive directives. The race condition occurs when one task executes a Receive directive and finds its receive queue empty; but before the task can exit, the other task sends it a message. The message is lost



because the Executive flushed the receiver task's receive queue when it decided to exit. This condition can be avoided if the sending task specifies a common event flag in the Send directive and the receiving task executes an Exit If specifying the same common event flag. If the event flag is set, the Exit If directive returns control to the issuing task, signaling that something has been sent.

2. A Fortran program that issues the Exit If call must first close all files by issuing Close calls. To avoid the time overhead involved in closing and reopening files, the task should first issue the appropriate test or Clear Event Flag directive. If the Directive Status Word indicates that the flag was not set, then the task can close all files and issue the call to Exit If.
3. On Exit, the Executive frees task resources. In particular, the Executive:
  - Detaches all attached devices
  - Flushes the AST queue and despecifies all specified ASTs
  - Flushes the receive and receive-by-reference queues
  - Flushes the clock queue for any outstanding Mark Time requests for the task
  - Closes all open files (files open for write access are locked)
  - Detaches all attached regions, except in the case of a fixed task
  - Runs down the task's I/O
  - Disconnects from interrupt vectors
  - Breaks the connection with any offspring tasks
  - Returns a success status (EX\$SUC) to any parent tasks
  - Frees the task's memory if the exiting task was not fixed
4. If the task exits, the Executive declares a significant event.

## EXIT\$\$

### 9.1.23 EXIT\$\$—Task Exit (\$\$ Form Recommended)

The Task Exit directive instructs the system to terminate the execution of the issuing task.

#### Fortran Call

See Note 5.

#### Macro Call

```
EXIT$$ [err]
```

err     error routine address

#### Macro Expansion

```
EXIT$$  ERR
MOV     (PC)+, -(SP)      ;PUSH DPB ONTO THE STACK
.BYTE   51., 1            ;EXIT$$ MACRO DIC, DPB SIZE=1 WORD
EMT     377                ;TRAP TO THE EXECUTIVE
JSR     PC,ERR            ;CALL ROUTINE "ERR"
```

#### Local Symbol Definitions

None

#### DSW Return Codes

IE.ADP       part of the DPB is out of the issuing task's address space

IE.SDP       DIC or DPB size is invalid

#### Notes

1. A return to the task occurs if, and only if, the directive is rejected. Therefore, no Branch on Carry Clear instruction is generated if an error routine address is given, since the return will only occur with carry set.
2. Exit causes a significant event to be declared.
3. On Exit, the Executive frees task resources. In particular, the Executive:
  - Detaches all attached devices
  - Flushes the AST queue and despecifies all specified ASTs
  - Flushes the receive and receive-by-reference queues

- Flushes the clock queue for any outstanding Mark Time requests for the task
  - Closes all open files (files open for write access are locked)
  - Detaches all attached regions, except in the case of a fixed task, where no detaching occurs
  - Runs down the task's I/O
  - Disconnects from interrupt vectors
  - Breaks the connection with any offspring tasks
  - Returns a success code (EX\$SUC) to any parent task
  - Frees the task's memory if the exiting task was not fixed
4. Because this directive requires only a 1-word DPB, the \$S form of the macro is recommended. It requires less space and executes with the same speed as that of the DIR\$ macro.
5. You can terminate Fortran tasks with the STOP statement or with CALL EXIT. CALL EXIT is a Fortran OTS routine that closes open files and performs other cleanup before it issues an EXIT\$\$ directive (or an EXST\$ directive in FORTRAN-77). Fortran tasks that terminate with the STOP statement result in a message being displayed on the task's TI:. This message includes task name (as it appears in the Active Task List), the statement causing the task to stop, and an optional character string specified in the STOP statement. Tasks that terminate with CALL EXIT do not display a termination message.

For example, a Fortran task containing the following statement:

```
20 STOP 'THIS FORTRAN TASK'
```

exits with the following message displayed on the task's TI: (TT0 in this example):

```
TT0 - STOP THIS FORTRAN TASK
```

## EXST\$

### 9.1.24 EXST\$—Exit With Status

The Exit With Status directive causes the issuing task to exit, passing a 16-bit status back to all tasks connected (by the Spawn, Connect, or Send, Request And Connect directive). If the issuing task has no connected tasks, then the directive simply performs a Task Exit. No format of the status word is enforced by the Executive; format conventions are a function of the cooperation between parent and offspring tasks. However, if an offspring task aborts for any reason, a status of EX\$SEV is returned to the parent task. This value is interpreted as a “severe error” by batch processors. Furthermore, if a task performs a normal exit with other tasks connected to it, a status of EX\$SUC (successful completion) is returned to all connected tasks.

#### Fortran Call

```
CALL EXST (istat)
```

istat            a 16-bit quantity to be returned to parent task

#### Macro Call

```
EXST$ status
```

status           a 16-bit quantity to be returned to parent task

#### Macro Expansion

```
EXST$        STWD
.BYTE        29.,2            ;EXST$ MACRO DIC, DPB SIZE=2 WORDS
.WORD        STWD            ;VALUE OF STATUS TO BE RETURNED
```

#### Local Symbol Definitions

E.XSTS        value of status to be returned (2)

#### DSW Return Codes

No status is returned if the directive is successfully completed since the directive causes the issuing task to exit.

IE.ADP        part of the DPB is out of the issuing task's address space  
 IE.SDP        DIC or DPB size is invalid

**Notes**

1. The executive does the following to free a task's resources on Exit:
  - Detaches all attached devices
  - Flushes the AST queue and despecifies all specified ASTs
  - Flushes the Receive and Receive-by-reference queues
  - Flushes the clock queue for any outstanding Mark Time requests for the task
  - Closes all open files (files open for write access are locked)
  - Detaches all attached regions except in the case of a fixed task
  - Runs down the task's I/O
  - Disconnects from interrupt vectors
  - Breaks the connection with any offspring tasks
  - Returns the specified exit status to any parent tasks
  - Frees the task's memory if the exiting task was not fixed
2. If the task exits, the executive declares a significant event.

## EXTK\$

### 9.1.25 EXTK\$—Extend Task

The Extend Task directive instructs the system to modify the size of the issuing task by a positive or negative increment of 32-word blocks. If the directive does not specify an increment value or specifies an increment value of zero, the Executive makes the issuing task's size equal to its installed size. The issuing task and cannot have any outstanding I/O when it issues the directive. The task must also be checkpointable to increase its size; if necessary, the Executive checkpoints the task, and then returns the task to memory with its size modified as directed.

The Executive does not change any current mapping assignments if the task has memory-resident overlays. However, if the task does not have memory-resident overlays, the Executive attempts to modify, by the specified number of 32-word blocks, the mapping of the task to its task region.

If the issuing task is checkpointable but has no preallocated checkpoint space available, a positive increment may require dynamic memory and extra space in a checkpoint file sufficient to contain the task.

There are several constraints on the size to which a task can extend itself using the Extend directive:

- A task that does not have memory-resident overlays cannot extend itself beyond 32K minus 32 words.
- A task that has preallocated checkpoint space in its task image file cannot extend itself beyond its installed size.
- A task that has memory-resident overlays cannot reduce its size below the highest window in the task partition.

### Fortran Call

```
CALL EXTTSK ([inc][,ids])
```

inc     a positive or negative number equal to the number of 32-word blocks by which the task size is to be extended or reduced

ids     directive status

### Macro Call

```
EXTK$ [inc]
```

inc     a positive or negative number equal to the number of 32-word blocks by which the task size is to be extended or reduced

**Macro Expansion**

```

EXTK$ 40
.BYTE 89.,3 ;EXTK$ MACRO DIC, DPB SIZE=3 WORDS
.WORD 40 ;EXTEND INCREMENT, 40(8) BLOCKS (1K
;WORDS)
.WORD 0 ;RESERVED WORD

```

**Local Symbol Definitions**

E.XTIN extend increment (2)

**DSW Return Codes**

IS.SUC successful completion

IE.UPN insufficient dynamic memory, or insufficient space in a checkpoint file

IE.ITS the issuing task is not running in a system controlled partition

IE.ALG the issuing task attempted to reduce its size to less than the size of its task header; or the task tried to increase its size beyond 32K words; or the task tried to increase its size to the extent that one virtual address window would overlap another; or the task has memory-resident overlays and it attempted to reduce its size below the highest window mapped to the task partition

IE.RSU other tasks are attached to this task partition

IE.IOP I/O is in progress for this task partition

IE.CKP the issuing task is not checkpointable and specified a positive integer

IE.NSW attempt to extend to larger than installed size (when checkpoint space is allocated in the task)

IE.ADP part of the DPB is out of the issuing task's address space

IE.SDP DIC or DPB size is invalid

## FEAT\$

### 9.1.26 FEAT\$ — Test for Specified System Feature

The Features directive tests for the presence of a specific system software or hardware option (such as floating point support or the presence of the Commercial Instruction Set).

#### Fortran Call

```
CALL FEAT (ISYM,[,ids])
```

isym            symbol for the specified system feature  
ids             directive status

#### Macro Call

```
FEAT$ sym
```

sym            symbol for the specified system feature

#### Macro Expansion

```
FEAT$  FE$POS  
.BYTE  177.,2      ;FEAT$ MACRO DIC, DPB SIZE=2 WORDS  
.WORD  FE$POS     ;Feature identifier
```

#### Local Symbol Definitions

F.EAF         feature identifier (2)

#### DSW Return Codes

IS.CLR        successful completion; feature not present  
IS.SET        successful completion; feature present  
IE.ADP        part of the DPB is out of the issuing task's address space  
IE.SDP        DIC or DPB size is invalid



Table 9-6  
System Feature Symbols

<i>Symbol</i>	<i>Meaning</i>
FE\$EXT	22-BIT EXTENDED MEMORY SUPPORT (BIT 1)
FE\$MUP	MULTIUSER PROTECTION SUPPORT
FE\$EXV	EXECUTIVE IS SUPPORTED TO 20K WORDS
FE\$DRV	LOADABLE DRIVER SUPPORT
FE\$PLA	PLAS SUPPORT
FE\$CAL	DYNAMIC CHECKPOINT SPACE ALLOCATION
FE\$PKT	PREALLOCATION OF I/O PACKETS
FE\$EXP	EXTEND TASK DIRECTIVE SUPPORT
FE\$LSI	PROCESSOR IS AN LSI-11
FE\$OFF	PARENT/OFFSPRING TASKING SUPPORT
FE\$FDT	FULL-DUPLEX TERMINAL DRIVER SUPPORT
FE\$X25	X.25 CEX IS LOADED
FE\$DYM	DYNAMIC MEMORY ALLOCATION SUPPORTED
FE\$CEX	COM EXEC IS LOADED
FE\$MXT	MCR EXIT AFTER EACH COMMAND MODE
FE\$NLG	LOGINS DISABLED - MULTIUSER SUPPORT
FE\$DAS	KERNEL DATA SPACE SUPPORTED (BIT 17 <sub>10</sub> )
FE\$LIB	SUPERVISOR MODE LIBRARIES SUPPORT
FE\$MP	SYSTEM SUPPORTS MULTIPROCESSING
FE\$EVT	SYSTEM SUPPORTS EVENT TRACE FEATURE
FE\$ACN	SYSTEM SUPPORTS CPU ACCOUNTING
FE\$SDW	SYSTEM SUPPORTS SHADOW RECORDING
FE\$POL	SYSTEM SUPPORTS SECONDARY POOLS
FE\$WND	SYSTEM SUPPORTS SECONDARY POOL FILE WINDOWS
FE\$DPR	SYSTEM HAS A SEPERATE DIRECTIVE PARTITION
FE\$IRR	INSTALL, RUN, AND REMOVE SUPPORT
FE\$GGF	GROUP GLOBAL EVENT FLAG SUPPORT
FE\$RAS	RECEIVE/SEND DATA PACKET SUPPORT
FE\$AHR	ALT. HEADER REFRESH AREA SUPPORT
FE\$RBN	ROUND ROBIN SCHEDULING SUPPORT
FE\$SWP	EXECUTIVE LEVEL DISK SWAPPING SUPPORT
FE\$STP	EVENT FLAG MASK IS IN THE TCB(1=YES)
FE\$CRA	SYSTEM SPONTANEOUSLY CRASHED(1=YES) (BIT 33 <sub>10</sub> )
FE\$XCR	SYSTEM CRASHED FROM XDT(1=YES)
FE\$EIS	SYSTEM REQUIRES EXTENDED INSTRUCTION SET

Table 9-6 (Cont.)

<i>Symbol</i>	<i>Meaning</i>
FE\$STM	SYSTEM HAS SET SYSTEM TIME DIRECTIVE
FE\$UDS	SYSTEM SUPPORTS USER DATA SPACE
FE\$PRO	SYSTEM SUPPORTS SECONDARY POOL PROTO TCBS
FE\$XHR	SYSTEM SUPPORTS EXTERNAL TASK HEADERS
FE\$AST	SYSTEM HAS AST SUPPORT
FE\$11S	RSX-11S SYSTEM
FE\$CLI	MULTIPLE CLI SUPPORT
FE\$TCM	SYSTEM HAS SEPERATE TERMINAL DRIVER POOL
FE\$PMN	SYSTEM SUPPORTS POOL MONINTORING
FE\$WAT	SYSTEM HAS WATCHDOG TIMER SUPPORT
FE\$RLK	SYSTEM SUPPORTS RMS RECORD LOCKING
FE\$SHF	SYSTEM SUPPORTS SHUFFLER TASK
FE\$CXD	COMM EXEC IS DEALLOCATED (NON-I/D ONLY) (BIT 49 <sub>10</sub> )
FE\$POS	SYSTEM IS A P/OS SYSTEM (1=YES)
HF\$UBM	PROCESSOR HAS UNIBUS MAP (1=YES) (BIT 1)
HF\$EIS	PROCESSOR HAS EXTENDED INSTRUCTION SET
HF\$CIS	PROCESSOR SUPPORTS COMMERCIAL INSTRUCTION SET
HF\$FPP	PROCESSOR HAS NO FLOATING POINT UNIT (1=YES)
HF\$NVR	XT NONVOLATILE RAM PRESENT (1=YES) (BIT 17 <sub>10</sub> )
HF\$INV	NON-VOLATILE RAM PRESENT (1=YES)
HF\$CLK	P/OS CLOCK IS PRESENT
HF\$ITF	INVALID TIME FORMAT IN NONVOLATILE RAM
HF\$BRG	P/OS BRIDGE MODULE PRESENT

**GDIR\$****9.1.27 GDIR\$—Get Default Directory**

The Get Default Directory directive retrieves the default directory string, returning it and the string length to a user-specified buffer.

**Fortran Call**

```
CALL GETDDS (mod,iens,ienssz,[irsize],[idsw])
```

mod	the modifier of the logical name within a table
iens	character array containing the equivalence name string
ienssz	size (in bytes) of the equivalence name string
irsize	buffer address of the returned equivalence string size
idsw	integer to receive the Directive Status Word

**Macro Call**

```
GDIR$ mod,ens,enssz,rsize
```

mod	the modifier of the logical name within a table
ens	buffer address of the equivalence name string
enssz	size (in bytes) of the equivalence name string
rsize	buffer address to which the size of the equivalence name string is returned

**Macro Expansion**

```
GDIR$  mod,ens,enssz,rsize
.BYTE  207.,6          ;GDIR$ MACRO DIC AND DPB SIZE
.BYTE  4               ;SUBFUNCTION CODE FOR GET DEFAULT
                        ;DIRECTORY
.BYTE  MOD             ;LOGICAL NAME MODIFIER
.WORD  0               ;RESERVED
.WORD  ENS             ;BUFFER ADDRESS OF EQUIVALENCE NAME
.WORD  ENSSZ          ;BYTES COUNT OF EQUIVALENCE STRING
.WORD  RSIZE          ;BUFFER ADDRESS FOR RETURNED EQUIVALENCE
                        ;STRING
```

**Local Symbol Definitions**

G.DENS	address of equivalence name buffer (2)
G.DESZ	byte count of equivalence string (2)
G.DFUN	subfunction code (1)

- G.DMOD      logical name modifier (1)
- G.DRSZ      buffer address for returned equivalence string (2)

**DSW Return Codes**

- IS.SUC      successful completion of service
- IE.RBS      the resulting equivalence name string is too large for the buffer to receive it
- IE.LNF      the specified logical name string was not found
- IE.IBS      the length of the logical or equivalence string is invalid; each string length must be greater than 0 but not greater than 255<sub>10</sub> characters
- IE.ADP      part of the DPB or user buffer is out of the issuing task's address space, or the user does not have proper access to that region
- IE.SDP      DIC or DPB size is invalid

**GLUN\$****9.1.28 GLUN\$—Get LUN Information**

The Get LUN Information directive instructs the system to fill a 6-word buffer with information about a physical device unit to which a LUN is assigned. If requests to the physical device unit have been redirected to another unit, the information returned will describe the effective assignment.

**Fortran Call**

CALL GETLUN (lun,dat[,ids])

lun	logical unit number
dat	a 6-word integer array to receive LUN information
ids	directive status

**Macro Call**

GLUN\$ lun,buf

lun	logical unit number
buf	address of 6-word buffer that will receive the LUN information

**Buffer Format**

Word 0	name of assigned device														
Word 1	unit number of assigned device and flags byte (flags byte equals 200 if the device driver is resident or 0 if the driver is not loaded)														
Word 2	first device characteristics word: <table> <tr> <td>Bit 0</td> <td>record-oriented device (DV.REC,1=yes)[FD.REC]<sup>8</sup></td> </tr> <tr> <td>Bit 1</td> <td>carriage-control device (DV.CCL,1=yes)[FD.CCL]</td> </tr> <tr> <td>Bit 2</td> <td>terminal device (DV.TTY,1=yes)[FD.TTY]</td> </tr> <tr> <td>Bit 3</td> <td>directory (file-structured) device (DV.DIR,1=yes)[FD.DIR]</td> </tr> <tr> <td>Bit 4</td> <td>reserved for future use</td> </tr> <tr> <td>Bit 5</td> <td>sequential device (DV.SQD,1=yes)[FD.SQD]</td> </tr> <tr> <td>Bit 6</td> <td>mass storage device (DV.MSD,1=yes)</td> </tr> </table>	Bit 0	record-oriented device (DV.REC,1=yes)[FD.REC] <sup>8</sup>	Bit 1	carriage-control device (DV.CCL,1=yes)[FD.CCL]	Bit 2	terminal device (DV.TTY,1=yes)[FD.TTY]	Bit 3	directory (file-structured) device (DV.DIR,1=yes)[FD.DIR]	Bit 4	reserved for future use	Bit 5	sequential device (DV.SQD,1=yes)[FD.SQD]	Bit 6	mass storage device (DV.MSD,1=yes)
Bit 0	record-oriented device (DV.REC,1=yes)[FD.REC] <sup>8</sup>														
Bit 1	carriage-control device (DV.CCL,1=yes)[FD.CCL]														
Bit 2	terminal device (DV.TTY,1=yes)[FD.TTY]														
Bit 3	directory (file-structured) device (DV.DIR,1=yes)[FD.DIR]														
Bit 4	reserved for future use														
Bit 5	sequential device (DV.SQD,1=yes)[FD.SQD]														
Bit 6	mass storage device (DV.MSD,1=yes)														

8. Bits with associated symbols have the symbols shown in square brackets. These symbols can be defined for use by a task by means of the FCSBT\$ macro.

Bit 7	reserved for future use
Bit 8	reserved for future use
Bit 9	unit software write-locked (DV.SWL,1=yes)
Bit 10	reserved for future use
Bit 11	reserved for future use
Bit 12	pseudo device (DV.PSE,1=yes)
Bit 13	device mountable as a communications channel (DV.COM,1=yes)
Bit 14	device mountable as a Files-11 device (DV.F11,1=yes)
Bit 15	device mountable (DV.MNT,1=yes)
Word 3	second device characteristics word
Word 4	third device characteristics word (words 3 and 4 are device driver specific)
Word 5	fourth device characteristics word (normally buffer-size)

### Macro Expansion

```

GLUN$ 7,LUNBUF
.BYTE 5,3          ;GLUN$ MACRO DIC, DPB SIZE=3 WORDS
.WORD 7           ;LOGICAL UNIT NUMBER 7
.WORD LUNBUF      ;ADDRESS OF 6-WORD BUFFER

```

### Local Symbol Definitions

G.LULU      logical unit number (2)  
G.LUBA      buffer address (2)

The following offsets are assigned relative to the start of the LUN information buffer:

G.LUNA      device name (2)  
G.LUNU      device unit number (1)  
G.LUFB      flags byte (1)  
G.LUCW      four device characteristics words (8)

### DSW Return Codes

IS.SUC      successful completion  
IE.ULN      unassigned LUN  
IE.ILU      invalid logical unit number

IE.ADP      part of the DPB or buffer is out of the issuing task's address space

IE.SDP      DIC or DPB size is invalid

**Note**

None

## GMCR\$

### 9.1.29 GMCR\$—Get Command Line

The Get Command Line directive instructs the system to transfer an 80-byte command line to the issuing task.

#### Fortran Call

```
CALL GETMCR (buf[,ids])
```

buf an 80-byte array to receive command line

ids directive status

#### Macro Call

```
GMCR$
```

#### Macro Expansion

```
GMCR$
.BYTE 127.,41. ;GMCR$ MACRO DIC, DPB SIZE=41. WORDS
.BLKW 40. ;80. CHARACTER MCR COMMAND LINE BUFFER
```

#### Local Symbol Definitions

G.MCRB command line buffer (80)

#### DSW Return Codes

+n	successful completion; n is the number of data bytes transferred (excluding the termination character). The termination character is, however, in the buffer
IE.AST	no command line exists for the issuing task, or the task has already issued one or more Get Command Line directives and has retrieved the entire command line
IE.ADP	part of the DPB is out of the issuing task's address space
IE.SDP	DIC or DPB size is invalid



**Notes**

1. The GMCR\$\$ form of the macro is not supplied, since the DPB receives the actual command line.
2. The system processes all lines to:
  - Convert tabs to a single space
  - Convert multiple spaces to a single space
  - Convert lowercase to uppercase
  - Remove all trailing blanks

The terminator (<RETURN> or <ESC>) is the last character in the line.

3. If the character before the terminator is a hyphen, there is at least one continuation line present. Therefore, you must issue another GMCR\$ directive to obtain the rest of the command line.

## GMCX\$

### 9.1.30 GMCX\$—Get Mapping Context

The Get Mapping Context directive causes the Executive to return a description of the current window-to-region mapping assignments. The returned description is in a form that enables the user to restore the mapping context through a series of Create Address Window directives. The macro argument specifies the address of a vector that contains one Window Definition Block (WDB) for each window block allocated in the task's header, plus a terminator word.

For each window block in the task's header, the Executive sets up a WDB in the vector as follows:

1. If the window block is unused (that is, if it does not correspond to an existing address window), the Executive does not record any information about that block in a WDB. Instead, the Executive uses the WDB to record information about the first block encountered that corresponds to an existing window. In this way, unused window blocks are ignored in the mapping context description returned by the Executive.
2. If a window block describes an existing unmapped address window, the Executive fills in the offsets W.NID, W.NAPR, W.NBAS, and W.NSIZ with information sufficient to recreate the window. The window status word W.NSTS is cleared.
3. If a window block describes an existing mapped window, the Executive fills in the offsets W.NAPR, W.NBAS, W.NSIZ, W.NRID, W.NOFF, W.NLEN, and W.NSTS with information sufficient to create and map the address window. WS.MAP is set in the status word (W.NSTS) and, if the window is mapped with write access, the bit WS.WRT is set as well.

Note that in no case does the Executive modify W.NSRB.

The terminator word, which follows the last WDB filled in, is a word equal to the negative of the total number of window blocks in the task's header. It is thereby possible to issue a TST or TSTB instruction to detect the last WDB used in the vector. The terminating word can also be used to determine the number of window blocks built into the task's header.

When Create Address Window directives are used to restore the mapping context, there is no guarantee that the same address window IDs will be used. The user must therefore be careful to use the latest window IDs returned from the Create Address Window directives.

#### Fortran Call

```
CALL GMCX (imcx[,ids])
```

**imcx** an integer array to receive the mapping context; the size of the array is  $8*n+1$  where  $n$  is the number of window blocks in the task's header; the maximum size is  $8*24+1=193$  words.

**ids** directive status

**Macro Call**

GMCX\$ wvec

wvec the address of a vector of n Window Definition Blocks, followed by a terminator word; n is the number of window blocks in the task's header

**Macro Expansion**

```
GMCX$  VECADR
.BYTE  113. ,2      ;GMCX$ MACRO DIC, DPB SIZE=2 WORDS
.WORD  VECADR      ;WDB VECTOR ADDRESS
```

Table 9-7  
Window Definition Block Parameters

<i>Input Parameters</i>		
None		
<i>Output Parameters</i>		
<i>Array Element</i>	<i>Offset</i>	<i>Description</i>
iwdb(1) bits 0-7	W.NID	ID of address window
iwdb(1) bits 8-15	W.NAPR	Base APR of the window
iwdb(2)	W.NBAS	Base virtual address of the window
iwdb(3)	W.NSIZ	Size, in 32-word blocks, of the window
iwdb(4)	W.NRID	ID of the mapped region, or no change if the window is unmapped
iwdb(5)	W.NOFF	Offset, in 32-word blocks, from the start of the region at which mapping begins, or no change if the window is unmapped
iwdb(6)	W.NLEN	Length, in 32-word blocks, of the area currently mapped within the region, or no change if the window is unmapped
iwdb(7)	W.NSTS	Bit settings <sup>9</sup> in the window status word (all 0 if the window is not mapped):
	<i>Bit</i>	<i>Definition</i>
	WS.MAP	1 if the window is mapped
	WS.WRT	1 if the window is mapped with write access

Note: The length mapped (W.NLEN) can be less than the size of the window (W.NSIZ) if the area from W.NOFF to the end of the partition is smaller than the window size.

9. If you are a Fortran programmer, refer to Section 7.5.2 to determine the bit values represented by the symbolic names described.

### Local Symbol Definitions

G.MCVA        address of the vector (wvec) containing the window definition blocks and terminator word (2)

### DSW Return Codes

IS.SUC        successful completion  
IE.ADP        address check of the DPB or the vector (wvec) failed  
IE.SDP        DIC or DPB size is invalid

### Note

None

**GPRT\$****9.1.31 GPRT\$—Get Partition Parameters**

The Get Partition Parameters directive instructs the system to fill an indicated 3-word buffer with partition parameters. If a partition is not specified, the partition of the issuing task is assumed.

**Fortran Call**

```
CALL GETPAR ([prt],buf[,ids])
```

prt    partition name  
buf    a 3-word integer array to receive partition parameters  
ids    directive status

**Macro Call**

```
GPRT$ [prt],buf
```

prt    partition name  
buf    address of a 3-word buffer

**Buffer Format**

Word 0      partition physical base address expressed as a multiple of 32 words (partitions are always aligned on 32-word boundaries); therefore, a partition starting at 40000<sub>8</sub> will have 400<sub>8</sub> returned in this word.  
Word 1      partition size expressed as a multiple of 32 words.  
Word 2      partition flags word; this word is returned equal to 0 to indicate a system-controlled partition, or equal to 1 to indicate a user-controlled partition.

**Macro Expansion**

```
GPRT$    ALPHA, DATBUF
.BYTE    65., 4                    ;GPRT$ DIC, DPB SIZE=4 WORDS
.RAD50   /ALPHA/                 ;PARTITION ''ALPHA''
.WORD    DATBUF                   ;ADDRESS OF 3-WORD BUFFER
```

**Local Symbol Definitions**

G.PRPN    partition name (4)  
G.PRBA    buffer address (2)

The following offsets are assigned relative to the start of the partition parameters buffer:

G.PRPB	partition physical base address expressed as an absolute 32-word block number (2)
G.PRPS	partition size expressed as a multiple of 32-word blocks (2)
G.PRFW	partition flags word (2)

### DSW Return Codes

Successful completion is indicated by a cleared Carry bit, and the starting address of the partition is returned in the DSW. The returned address is virtual and is always zero if it is not the task partition. Unsuccessful completion is indicated by a set Carry bit and one of the following codes in the DSW:

IE.INS	specified partition not in system
IE.ADP	part of the DPB or buffer is out of the issuing task's address space
IE.SDP	DIC or DPB size is invalid

### Notes

1. A variation of this directive exists called Get Region Parameters. When the first word of the 2-word partition name is 0, the Executive interprets the second word of the partition name as a region ID. If the 2-word name is 0,0, it refers to the task region of the issuing task.
2. Omission of the partition-name argument returns parameters for the issuing task's unnamed subpartition, not for the system-controlled partition.

**GREG\$****9.1.32 GREG\$—Get Region Parameters**

The Get Region Parameters directive instructs the Executive to fill an indicated 3-word buffer with region parameters. If a region is not specified, the task region of the issuing task is assumed.

This directive is a variation of the Get Partition Parameters directive.

**Fortran Call**

```
CALL GETREG ([rid],buf[,ids])
```

rid     region id  
buf     a 3-word integer array to receive region parameters  
ids     directive status

**Macro Call**

```
GREG$ [rid],buf
```

rid     region ID  
buf     address of a 3-word buffer

**Buffer Format**

Word 0     region base address expressed as a multiple of 32 words (regions are always aligned on 32-word boundaries); thus, a region starting at 1000<sub>8</sub> will have 10<sub>8</sub> returned in this word  
Word 1     region size expressed as a multiple of 32 words  
Word 2     region flags word; this word is returned equal to 0 if the region resides in a system-controlled partition, or equal to 1 if the region resides in a user-controlled partition

**Macro Expansion**

```
GREG$  RID,DATBUF
.BYTE  65.,4           ;GREG$ MACRO DIC, DPB SIZE=4 WORDS
.WORD  0              ;WORD THAT DISTINGUISHES GREG$
                          ;FROM GPRT$
.WORD  RID            ;REGION ID
.WORD  DATBUF         ;ADDRESS OF 3-WORD BUFFER
```

**Local Symbol Definitions**

G.RGID        region ID (2)  
G.RGBA        buffer address

The following offsets are assigned relative to the start of the region parameters buffer:

G.RGRB        region base address expressed as an absolute 32-word block  
                  number (2)  
G.RGRS        region size expressed as a multiple of 32-word blocks (2)  
G.RGFW        region flags word (2)

**DSW Return Codes**

Successful completion is indicated by carry clear, and the starting address of the region is returned in the DSW. The returned address is virtual and is always zero if it is not the task region. Unsuccessful completion is indicated by carry set and one of the following codes in the DSW:

IE.NVR        invalid region ID  
IE.ADP        part of the DPB or buffer is out of the issuing task's address  
                  space  
IE.SDP        DIC or DPB size is invalid



**GTIM\$****9.1.33 GTIM\$—Get Time Parameters**

The Get Time Parameters directive instructs the system to fill an indicated 8-word buffer with the current time parameters. All time parameters are delivered as binary numbers. The value ranges (in decimal) are shown below in the buffer format.

**Fortran Call**

```
CALL GETTIM (ibfp[,ids])
```

ibfp an 8-word integer array

**Macro Call**

```
GTIM$ buf
```

buf address of 8-word buffer

**Buffer Format**

Word 0	year (since 1900)
Word 1	month (1-12)
Word 2	day (1-31)
Word 3	hour (0-23)
Word 4	minute (0-59)
Word 5	second (0-59)
Word 6	tick of second (Fixed rate of 64.)
Word 7	ticks per second (Fixed rate of 64.)

**Macro Expansion**

```
GTIM$  DATBUF
.BYTE  61.,2      ;GTIM$ DIC, DPB SIZE=2 WORDS
.WORD  DATBUF    ;ADDRESS OF 8.-WORD BUFFER
```

### Local Symbol Definitions

G.TIBA            buffer address (2)

The following offsets are assigned relative to the start of the time parameters buffer:

G.TIYR            year (2)

G.TIMO            month (2)

G.TIDA            day (2)

G.TIHR            hour (2)

G.TIMI            minute (2)

G.TISC            second (2)

G.TICT            clock Tick of Second (2)

G.TICP            clock Ticks per Second (2)

### DSW Return Codes

IS.SUC            successful completion

IE.ADP            part of the DPB or buffer is out of the issuing task's address space

IE.SDP            DIC or DPB size is invalid

### Note

1. The format of the time buffer is compatible with that of the buffers used with the Set System Time directive.

**GTSK\$****9.1.34 GTSK\$—Get Task Parameters**

The Get Task Parameters directive instructs the system to fill an indicated 16-word buffer with parameters relating to the issuing task.

**Fortran Call**

```
CALL GETTSK (buf[,ids])
```

buf a 16-word integer array to receive the task parameters

ids directive status

**Macro Call**

```
GTSK$ buf
```

buf address of a 16-word buffer

**Buffer Format**

Word 0	issuing task's name in Radix-50 (first half)
Word 1	issuing task's name in Radix-50 (second half)
Word 2	partition name in Radix-50 (first half)
Word 3	partition name in Radix-50 (second half)
Word 4	undefined in the system
Word 5	undefined in the system
Word 6	run priority
Word 7	User Identification Code (UIC) of issuing task (the task's default UIC) <sup>10</sup>
Word 10	number of logical I/O units (LUNs)
Word 11	undefined in the system
Word 12	undefined in the system
Word 13	(address of task SST vector tables) <sup>11</sup>
Word 14	(size of task SST vector table in words) <sup>11</sup>
Word 15	size (in bytes) either of task's address window 0 in mapped systems, or of task's partition in unmapped system (equivalent to partition size)
Word 16	system on which task is running: 11 the P/OS system always returns this code
Word 17	protection UIC

10. See note in RQST\$ description on contents of words 07 and 17.

11. Words 13 and 14 will contain valid data if word 14 is not zero. If word 14 is zero, the contents of word 13 are meaningless.

**Macro Expansion**

```

GTSK$   DATBUF
.BYTE   63.,2           ;GTSK$ MACRO DIC, DPB SIZE = 2 WORDS
.WORD   DATBUF         ;ADDRESS OF 16-WORD BUFFER

```

**Local Symbol Definitions**

G.TSBA        buffer address (2)

The following offsets are assigned relative to the task parameter buffer:

G.TSTN        task name (4)  
G.TSPN        partition name (4)  
G.TSPR        priority (2)  
G.TSGC        UIC group code (1)  
G.TSPC        UIC member code (1)  
G.TSNL        number of logical units (2)  
G.TSVA        task's SST vector address (2)  
G.TSVL        task's SST vector length in words (2)  
G.TSTS        task size (2)  
G.TSSY        system on which task is running (2)  
G.TSDU        protection UIC (2)

**DSW Return Codes**

IS.SUC        successful completion  
IE.ADP        part of the DPB or buffer is out of the issuing task's address space  
IE.SDP        DIC or DPB is invalid

**MAP\$****9.1.35 MAP\$—Map Address Window**

The Map Address Window directive maps an existing window to an attached region. The mapping begins at a specified offset from the start of the region. If the window is already mapped elsewhere, the Executive unmaps it before carrying out the mapping assignment described in the directive.

For the mapping assignment, a task can specify any length that is less than or equal to both:

- The window size specified when the window was created
- The length remaining between the specified offset within the region and the end of the region

A task must be attached with write access to a region in order to map to it with write access. To map to a region with read-only access, the task must be attached with either read or write access.

If W.NLEN is set to 0, the length defaults to either the window size or the length remaining in the region, whichever is smaller. (Since the Executive returns the actual length mapped as an output parameter, the task must clear that parameter in the WDB before issuing the directive each time it wants to default the length of the map.)

The values that can be assigned to W.NOFF depend on the setting of bit WS.64B in the window status word (W.NSTS):

- If WS.64B = 0, the offset specified in W.NOFF must represent a multiple of 256 words (512 bytes). Because the value of W.NOFF is expressed in units of 32-word blocks, the value must be a multiple of 8.
- If WS.64B = 1, the task can align on 32-word boundaries; the programmer can therefore specify any offset within the region.

**Fortran Call**

```
CALL MAP (iwdb[,ids])
```

iwdb    an 8-word integer array containing a Window Definition Block (see Section 7.5.2.2)

ids     directive status

**Macro Call**

```
MAP$ wdb
```

wdb    window Definition Block address

**Macro Expansion**

```

MAP$      WDBADR
.BYTE     121.,2           ;MAP$ MACRO DIC, DPB SIZE=2 WORDS
.WORD     WDBADR          ;WDB ADDRESS

```

Table 9-8  
Window Definition Block Parameters

*Input Parameters*

<i>Array Element</i>	<i>Offset</i>	<i>Description</i>
iwdb(1) bits 0-7	W.NID	ID of the window to be mapped.
iwdb(4)	W.NRID	ID of the region to which the window is to be mapped, or 0 if the task region is to be mapped.
iwdb(5)	W.NOFF	Offset, in 32-word blocks, within the region at which mapping is to begin. Note that if WS.64B in the window status word equals 0, the value specified must be a multiple of 8.
iwdb(6)	W.NLEN	Length, in 32-word blocks, within the region to be mapped, or 0 if the length is to default to either the size of the window or the space remaining in the region from the specified offset, whichever is smaller.
iwdb(7)	W.NSTS	Bit settings <sup>12</sup> in the window status word:
		<i>Bit</i> <i>Definition</i>
		WS.WRT                  1 if write access is desired
		WS.64B                  0 for 256-word (512-byte) alignment, or 1 for 32-word (64-byte) alignment

*Output Parameters*

iwdb(6)	W.NLEN	Length of the area within the region actually mapped by the window
iwdb(7)	W.NSTS	Bit settings <sup>13</sup> in the window status word:
	WS.UNM	1 if the window was unmapped first

12. If you are a Fortran programmer, refer to Section 7.5.2 to determine the bit values represented by the symbolic names described.

13. If you are a Fortran programmer, refer to Section 7.5.2 to determine the bit values represented by the symbolic names described.

**Local Symbol Definitions**

M.APBA window Definition Block address (2)

**DSW Return Codes**

IS.SUC	successful completion
IE.PRI	privilege violation
IE.NVR	invalid region ID
IE.NVW	invalid address window ID
IE.ALG	task specified an invalid region offset and length combination in the Window Definition Block parameters; or WS.64B = 0 and the value of W.NOFF is not a multiple of 8
IE.HWR	region had a parity error or a load failure
IE.ITS	WS.RES was set and region is not resident
IE.ADP	part of the DPB or WDB is out of the issuing task's address space
IE.SDP	DIC or DPB size is invalid

**Notes**

1. When the Map Address Window directive is issued, the task may be blocked until the region is loaded.
2. Bit WS.RES in word W.NSTS of the Window Definition Block, when set, specifies that the region should be mapped only if the region is resident.

## MRKT\$

### 9.1.36 MRKT\$—Mark Time

The Mark Time directive instructs the system to declare a significant event after an indicated time interval. The interval begins when the task issues the directive; however, task execution continues during the interval. If an event flag is specified, the flag is cleared when the directive is issued, and set when the significant event occurs. If an AST entry point address is specified, an AST (see Section 5.3.3) occurs at the time of the significant event. When the AST occurs, the task's PS, PC, directive status, Wait For mask words, and the event flag number specified in the directive are pushed onto the issuing task's stack. If neither an event flag number nor an AST service entry point is specified, the significant event still occurs after the indicated time interval. (See Notes.)

#### Fortran Calls

CALL MARK (efn,tmg,tnt[,ids])

efn    event flag number  
 tmg    time interval magnitude (see Note 5)  
 tnt    time interval unit (see Note 5)  
 ids    directive status

The ISA standard call for delaying a task for a specified time interval is also provided:

CALL WAIT (tmg,tnt[,ids])

tmg    time interval magnitude (see Note 5)  
 tnt    time interval unit (see Note 5)  
 ids    directive status

#### Macro Call

MRKT\$ [efn],tmg,tnt[,ast]

efn    event flag number  
 tmg    time interval magnitude (see Note 6)  
 tnt    time interval unit (see Note 6)  
 ast    AST entry point address



**Macro Expansion**

```

MRKT$  52.,30.,2,MRKAST
.BYTE  23.,5           ;MRKT$ MACRO DIC, DPB SIZE=5 WORDS
.WORD  52.            ;EVENT FLAG NUMBER 52.
.WORD  30.            ;TIME MAGNITUDE=30.
.WORD  2              ;TIME UNIT=SECONDS
.WORD  MRKAST         ;ADDRESS OF MARK TIME AST ROUTINE

```

**Local Symbol Definitions**

```

M.KTEF      event flag (2)
M.KTMG      time magnitude (2)
M.KTUN      time unit (2)
M.KTAE      AST entry point address (2)

```

**DSW Return Codes**

For CALL MARK and MRKT\$

```

IS.SUC      successful completion
IE.UPN      insufficient dynamic memory
IE.ITI      invalid time parameter
IE.IEF      invalid event flag number (EFN<0 or EFN>64)
IE.ADP      part of the DPB is out of the issuing task's address space
IE.SDP      DIC or DPB size is invalid

```

For CALL WAIT

The system provides the following positive error codes to be returned for ISA calls:

```

1          successful completion
2          insufficient dynamic storage
3          specified task not installed
94         invalid time parameters
98         invalid event flag number
99         part of DPB out of task's range
100       DIC or DPB size invalid

```

**Notes**

1. Mark Time requires dynamic memory for the clock queue entry.
2. If an AST entry point address is specified, the AST service routine is entered with the task's stack in the following state:

SP+10	event flag mask word <sup>14</sup>
SP+06	PS of task prior to AST
SP+04	PC of task prior to AST
SP+02	DSW of task prior to AST
SP+00	event flag number or zero (if none was specified in the Mark Time directive)

The event flag number must be removed from the task's stack before an AST Service Exit directive is executed.

3. If the directive is rejected, the specified event flag is not guaranteed to be cleared or set. Consequently, if the task indiscriminately executes a Wait For directive and the Mark Time directive is rejected, the task may wait indefinitely. Care should always be taken to ensure that the directive was successfully completed.
4. If a task issues a Mark Time directive that specifies a common event flag and then exits before the indicated time has elapsed, the event flag is not set.
5. The Executive returns the code IE.ITI (or 94) in the Directive Status Word if the directive specifies an invalid time parameter. The time parameter consists of two components: the time interval magnitude and the time interval unit, represented by the arguments tmg and tnt, respectively.

A legal magnitude value (tmg) is related to the value assigned to the time interval unit (tnt). The unit values are encoded as follows:

For an ISA Fortran call (CALL WAIT):

- 0 ticks. A tick occurs for each clock interrupt at a rate of 64(10) ticks per second
- 1 milliseconds. The subroutine converts the specified magnitude to the equivalent number of system clock ticks. On systems with line frequency clocks, millisecond Mark Time requests can only be approximations.

For all other Fortran and macro calls:

- 1 ticks. A tick occurs for each clock interrupt at a rate of 64(10) ticks per second.

---

14. The event flag mask word preserves the Wait For condition of a task prior to AST entry. A task can, after an AST, return to a Wait For state. Because these flags and the other stack data are in the use task, they can be modified. Such modification is strongly discouraged, however, since the task can easily fault on obscure conditions. For example, clearing the mask word results in a permanent Wait For state.

For both types of Fortran calls and all macro calls:

- 2 seconds
- 3 minutes
- 4 hours

The magnitude (tmg) is the number of units to be clocked. The following list describes the magnitude values that are valid for each type of unit. In no case can the value of tmg exceed 24 hours. The list applies to both Fortran and macro calls.

If  $tnt = 0, 1, \text{ or } 2$ , tmg can be any positive value with a maximum of 15 bits.

If  $tnt = 3$ , tmg can have a maximum value of 1440(10).

If  $tnt = 4$ , tmg can have a maximum value of 24(10).

6. The minimum time interval is one tick. If you specify a time interval of zero, it will be converted to one tick.

## QIO\$

### 9.1.37 QIO\$—Queue I/O Request

The Queue I/O Request directive instructs the system to place an I/O request for an indicated physical device unit into a queue of priority-ordered requests for that device unit. The physical device unit is specified as a logical unit number (LUN) assigned to the device.

The Executive declares a significant event when the I/O transfer completes. If the directive call specifies an event flag, the Executive clears the flag when the request is queued and sets the flag when the significant event occurs.

The I/O status block is also cleared when the request is queued and is set to the final I/O status when the I/O request is complete. If an AST service routine entry point address is specified, the AST occurs upon I/O completion, and the task's Wait For mask word, PS, PC, DSW, and the address of the I/O status block are pushed onto the task's stack.

The description below deals solely with the Executive directive. (See Notes.)

#### Fortran Call

```
CALL QIO (fnc,lun,[efn],[pri],[isb],[prl],[ids])
```

fnc    \*I/O function code (see Appendix C)

lun    logical unit number

efn    event flag number

pri    priority; ignored, but must be present

isb    a 2-word integer array to receive final I/O status

prl    a 6-word integer array containing device-dependent parameters to be placed in parameter words 1 through 6 of the DPB. Fill in this array by using the GETADR routine (see Section 3.4.1.4)

ids    directive status

#### Macro Call

```
QIO$ fnc,lun,[efn],[pri],[isb],[ast],[prl]
```

fnc    I/O function code (see Appendix C)

lun    logical unit number

efn    event flag number

pri    priority; ignored, but must be present

isb    address of I/O status block

ast    address of entry point of AST service routine

prl    parameter list of the form <P1,...P6>

**Macro Expansion**

```

QIO$      IO.RVB,7,52.,,IOSTAT,IOAST,<IOBUFR,512.>
.BYTE     1,12.           ;QIO$ MACRO DIC, DPB SIZE=12
.WORD     IO.RVB         ;FUNCTION=READ VIRTUAL BLOCK
.WORD     7              ;LOGICAL UNIT NUMBER 7
.BYTE     52.,0         ;EFN 52., PRIORITY IGNORED
.WORD     IOSTAT        ;ADDRESS OF 2-WORD I/O STATUS BLOCK
.WORD     IOAST         ;ADDRESS OF I/O AST ROUTINE
.WORD     IOBUFR        ;ADDRESS OF DATA BUFFER
.WORD     512.          ;BYTE COUNT=512.
.WORD     0              ;ADDITIONAL PARAMETERS...
.WORD     0              ;...NOT USED IN...
.WORD     0              ;...THIS PARTICULAR...
.WORD     0              ;...INVOCATION OF QUEUE I/O

```

**Local Symbol Definitions**

```

Q.IOFN      I/O function code (2)
Q.IOLU      logical unit number (2)
Q.IOEF      event flag number (1)
Q.IOPR      priority (1)
Q.IOSB      address of I/O status block (2)
Q.IOAE      address of I/O done AST entry point (2)
Q.IOPL      parameter list (6 words) (12)

```

**DSW Return Codes**

```

IS.SUC      successful completion
IE.UPN      insufficient dynamic memory
IE.ULN      unassigned LUN
IE.HWR      device driver not loaded
IE.PRI      task other than despooler attempted a write logical block
              operation.
IE.ILU      invalid LUN
IE.IEF      invalid event flag number (EFN<0 or EFN>64)
IE.ADP      part of the DPB or I/O status block is out of the issuing task's
              address space
IE.SDP      DIC or DPB size is invalid

```

**Notes**

1. If the directive call specifies an AST entry point address, the task enters the AST service routine with its stack in the following state:

SP+10	event flag mask word
SP+06	PS of task prior to AST
SP+04	PC of task prior to AST
SP+02	DSW of task prior to AST
SP+00	address of I/O status block, or zero, if none was specified in the QIO directive

The address of the I/O status block, which is a trap-dependent parameter, must be removed from the task's stack before an AST Service Exit directive is executed.

2. If the directive is rejected, the specified event flag is not guaranteed to be cleared or set. Consequently, if the task indiscriminately executes a Wait For or Stop For directive and the QIO directive is rejected, the task may wait indefinitely. Care should always be taken to ensure that the directive was successfully completed.
3. Tasks or regions cannot normally be checkpointed with I/O outstanding for two reasons:
  - If the QIO directive results in a data transfer, the data transfers directly to or from the user-specified buffer.
  - If an I/O status block address is specified, the directive status is returned directly to the I/O status block.

The Executive waits until a task has no outstanding I/O before initiating checkpointing in all cases except the one described below.

Drivers that buffer I/O check for the following conditions for a task:

- That the task is checkpointable
- That checkpointing is enabled

If those two conditions are met, the driver and/or the Executive buffers the I/O request internally and the task is checkpointable with this outstanding I/O. If the task also entered a Wait For state when the I/O was issued (see the QIOW\$ directive) or subsequently enters a Wait For state, the task is stopped. Any competing task waiting to be loaded into the partition can checkpoint the stopped task, regardless of priority. If the stopped task is checkpointed, the executive does not bring it back into memory until the stopped state is terminated by completion of buffered I/O or satisfaction of the Wait For condition.

Not all drivers buffer I/O requests. The terminal driver is an example of one that does.

4. Any task that is linked to a common (read-only) area can issue QIO write requests from that area.

**QIOW\$****9.1.38 QIOW\$—Queue I/O Request and Wait**

The Queue I/O Request And Wait directive is identical to the Queue I/O Request in all but one respect. If the Wait variation of the directive specifies an event flag, the Executive automatically effects a Wait For Single Event Flag directive. If an event flag is not specified, however, the Executive treats the directive as if it were a simple Queue I/O Request.

The following description lists the Fortran and macro calls with the associated parameters, as well as the macro expansion. Consult the description of Queue I/O Request for a definition of the parameters, the local symbol definitions, the DSW return codes, and explanatory notes.

**Fortran Call**

```
CALL WTQIO (fnc,lun,[efn],[pri],[isb],[prl][,ids])
```

fnc	I/O function code (see Appendix C)
lun	logical unit number
efn	event flag number
pri	priority; ignored, but must be present
isb	a 2-word integer array to receive final I/O status
prl	a 6-word integer array containing device-dependent parameters to be placed in parameter words 1 through 6 of the DPB
ids	directive status

**Macro Call**

```
QIOW$ fnc,lun,[efn],[pri],[isb],[ast][,prl]
```

fnc	I/O function code (see Appendix C)
lun	logical unit number
efn	event flag number
pri	priority; ignored, but must be present
isb	address of I/O status block
ast	address of entry point of AST service routine
prl	parameter list of the form <P1,...P6>

**Macro Expansion**

```

QIOW$  IO.RVB,7,52.,,IOSTAT,IOAST,<IOBUFR,512.>
.BYTE  3,12.           ;QIOW$ MACRO DIC, DPB SIZE=12.
WORD   IO.RVB         ;FUNCTION=READ VIRTUAL BLOCK
.WORD  7              ;LOGICAL UNIT NUMBER 7
.BYTE  52.,0          ;EFN 52., PRIORITY IGNORED
.WORD  IOSTAT         ;ADDRESS OF 2-WORD I/O STATUS BLOCK
.WORD  IOAST          ;ADDRESS OF I/O AST ROUTINE
.WORD  IOBUFR         ;ADDRESS OF DATA BUFFER
.WORD  512.           ;BYTE COUNT=512.
.WORD  0              ;ADDITIONAL PARAMETERS...
.WORD  0              ;...NOT USED IN...
.WORD  0              ;...THIS PARTICULAR...
.WORD  0              ;...INVOCATION OF QUEUE I/O

```



**RCST\$****9.1.39 RCST\$—Receive Data Or Stop**

The Receive Data Or Stop directive instructs the system to dequeue a 13-word data block for the issuing task; the data block was queued for the task with a Send Data Directive or a Send, Request And Connect directive.

A 2-word task name of the sender (in Radix-50 format) and the 13-word data block are returned in an indicated 15-word buffer. The task name is contained in the first two words of the buffer.

If no data has been sent, the issuing task is stopped. In this case, the sender task is expected to issue an Unstop directive after sending data. A success status code of IS.SUC indicates that a packet has been received. A success status code of IS.SET indicates that the task was stopped and has been unstopped. The directive must then be reissued to retrieve the packet.

When a slave task issues the Receive Data or Stop directive, it assumes the UIC and TI: terminal of the task that sent the data.

**Fortran Call**

CALL RCST ([rtname],ibuf[,ids])

rtname	sender task name (if not specified, data may be received from any task)
ibuf	address of 15-word buffer to receive the sender task name and data
ids	integer to receive the Directive Status Word

**Macro Call**

RCST\$ [tname],buf

tname	sender task name (If not specified, data may be received from any task)
buf	address of 15-word buffer to receive the sender task name and data

**Macro Expansion**

RCST\$	ALPHA, TSKBUF	
.BYTE	139., 4	;RCST\$ MACRO DIC, DPB SIZE=4 WORDS
.RAD50	ALPHA	;DATA SENDER TASK NAME
.WORD	TSKBUF	;BUFFER ADDRESS

### Local Symbol Definitions

R.CSTN      task name (4)  
R.CSBF      buffer address (2)

### DSW Return Codes

IS.SUC      successful completion  
IS.SET      no data was received and task was stopped (note that the task  
              must be Unstopped before it can see this status)  
IE.AST      the issuing task is at AST state  
IE.ADP      part of the DPB is out of the issuing task's address space  
IE.SDP      DIC or DPB size is invalid

### Note

1. The Receive Data Or Stop directive is treated as a 13<sub>10</sub> word Variable Receive Data Or Stop directive.

**RCVD\$****9.1.40 RCVD\$—Receive Data**

The Receive Data directive instructs the system to dequeue a 13-word data block for the issuing task; the data block has been queued (FIFO) for the task by a Send Data Directive.

A 2-word sender task name (in Radix-50 form) and the 13-word data block are returned in an indicated 15-word buffer, with the task name in the first two words.

When a slave task issues the Receive Data directive, it assumes the UIC and TI: terminal of the task that sent the data.

**Fortran Call**

```
CALL RECEIV ([tsk],buf[,ids])
```

tsk     sender task name (if not specified, data may be received from any task)  
 buf     a 15-word integer array for received data  
 ids     directive status

**Macro Call**

```
RCVD$ [tsk],buf
```

tsk     sender task name (if not specified, data may be received from any task)  
 buf     address of 15-word buffer

**Macro Expansion**

```
RCVD$     ALPHA,DATBUF     ;TASK NAME AND BUFFER ADDRESS
.BYTE     75.,4             ;RCVD$ MACRO DIC, DPB SIZE=4 WORDS
.RAD50    /ALPHA/           ;SENDER TASK NAME
.WORD     DATBUF            ;ADDRESS OF 15.-WORD BUFFER
```

**Local Symbol Definitions**

R.VDTN     sender task name (4)  
 R.VDBA     buffer address (2)

**DSW Return Codes**

IS.SUC     successful completion  
 IE.ITS     no data currently queued

IE.ADP	part of the DPB or buffer is out of the issuing task's address space
IE.SDP	DIC or DPB size is invalid

**Notes**

1. The Receive Data directive is treated as a  $13_{10}$  word Variable Receive Data directive.
2. If the sending task specifies a common event flag in the Send Data directive, the receiving task may use that event flag for synchronization. However, between the time that the receiver issues this directive and the time the receiver issues its next instruction, the sender can send data and set the event flag. If the next instruction is an Exit directive, any data sent during this time will be lost because the Executive flushes the task's receive list as part of exit processing. Therefore, use the Exit If directive or the Receive Data or Exit directive in order to avoid the race condition.

**RCVX\$****9.1.41 RCVX\$—Receive Data Or Exit**

The Receive Data Or Exit directive instructs the system to dequeue a 13-word data block for the issuing task; the data block has been queued (FIFO) for the task by a Send Data directive.

A 2-word sender task name (in Radix-50 form) and the 13-word data block are returned in an indicated 15-word buffer, with the task name in the first two words.

If no data has been sent, a task exit occurs. To prevent the possible loss of Send packets, the user should not rely on I/O rundown to take care of any outstanding I/O or open files; the task should assume this responsibility.

When a slave task issues the Receive Data Or Exit directive, it assumes the UIC and TI: of the task that sent the data. (See Notes.)

**Fortran Call**

```
CALL RECOEX ([tsk],buf[,ids])
```

tsk sender task name (if not specified, data may be received from any task)  
 buf a 15-word integer array for received data  
 ids directive status

**Macro Call**

```
RCVX$ [tsk],buf
```

tsk sender task name (if not specified, data may be received from any task)  
 buf address of 15-word buffer

**Macro Expansion**

```
RCVX$    ALPHA,DATBUF    ;TASK NAME AND BUFFER ADDRESS
.BYTE    77.,4           ;RCVX$ MACRO DIC, DPB SIZE=4 WORDS
.RAD50   /ALPHA/        ;SENDER TASK NAME
.WORD    DATBUF         ;ADDRESS OF 15.-WORD BUFFER
```

**Local Symbol Definitions**

R.VXTN sender task name (4)  
 R.VXBA buffer address (2)

**DSW Return Codes**

IS.SUC	successful completion
IE.ADP	part of the DPB or buffer is out of the issuing task's address space
IE.SDP	DIC or DPB size is invalid

**Notes**

1. A Fortran program that issues the RECOEX call must first close all files by issuing CLOSE calls.

To avoid the time overhead involved in the closing and reopening of files, the task should first issue the RECEIV call. If the directive status indicates that no data were received, then the task can close all files and issue the call to RECOEX. The following example illustrates the same overhead saving in MACRO:

```

RCVBUF: .BLKW 15. ; Receive buffer
START: RCVX$C ,RCVBUF ; Attempt to receive message
      CALL OPEN ; call user subroutine to
              ; open files.

PROC: .
      .
      Process packet of data
      .
      .
      RCVD$C ,RCVBUF ; Attempt to receive another
                  ; message
      BCC PROC ; If CC successful receive
      CALL CLOSE ; call user subroutine to
                  ; close files
                  ; and prepare for possible
                  ; task exit
      JMP START ; Make one last attempt
                  ; at receiving

```

2. If no data have been sent, that is, if no Send Data directive has been issued, the task exits. Send packets may be lost if a task exits with outstanding I/O or open files (see third paragraph of this section).
3. The Receive Data Or Exit directive is useful in avoiding a possible race condition that can occur between two tasks communicating by the Send and Receive directives. The race condition occurs when one task executes a Receive directive and finds its receive queue empty; but before the task can exit, the other task sends it a message. The message is lost because the Executive flushes the receiving task's receive queue when it exits. This condition can be avoided by the receiving task's executing a Receive Data Or Exit directive. If the receive queue is found to be empty, a task exit occurs before the other task can send any data; thus, no loss of data can occur.

4. On Exit, the Executive frees task resources. In particular, the Executive:
  - Detaches all attached devices
  - Flushes the AST queue and despecifies all specified ASTs
  - Flushes the receive and receive-by-reference queues
  - Flushes the clock queue for outstanding Mark Time requests for the task
  - Closes all open files (files open for write access are locked)
  - Detaches all attached regions except in the case of a fixed task, where no detaching occurs
  - Runs down the task's I/O
  - Disconnects from interrupt vectors
  - Returns a success status (EX\$SUC) to any parent tasks
  - Breaks the connection with any offspring tasks
  - Frees the task's memory if the exiting task was not fixed
5. If the task exits, the Executive declares a significant event.
6. The Receive Data Or Exit directive is treated as a 13-word Variable Receive Data Or Exit directive.

## RDAF\$

### 9.1.42 RDAF\$—Read All Event Flags

The Read All Event Flags directive instructs the system to read all 64 event flags for the issuing task and record their polarity in a 64-bit (4-word) buffer.

#### Fortran Call

A Fortran task can read only one event flag. The call is:

```
CALL READEF (efn[,ids])
```

efn event flag number

ids directive status

The Executive returns the status codes IS.SET (+02) and IS.CLR (00) for Fortran calls in order to report event flag polarity.

#### Macro Call

```
RDAF$ buf
```

#### Buffer Format

Word 0	task Local Flags 1-16
Word 1	task Local Flags 17-32
Word 2	task Common Flags 33-48
Word 3	task Common Flags 49-64

#### Macro Expansion

```
RDAF$    FLGBUF
.BYTE    39.,2      ;RDAF$ MACRO DIC, DPB SIZE=2 WORDS
.WORD    FLGBUF    ;ADDRESS OF 4-WORD BUFFER
```

#### Local Symbol Definitions

R.DABA buffer address (2)

#### DSW Return Codes

IS.SUC	successful completion
IE.ADP	part of the DPB or buffer is out of the issuing task's address space
IE.SDP	DIC or DPB size is invalid



**RDEF\$****9.1.43 RDEF\$—Read Event Flag**

The Read Event Flag directive tests an indicated event flag and reports its polarity in the DSW.

**Fortran Call**

```
CALL READEF (iefn[,ids])
```

iefn integer containing an Event Flag Number

ids integer variable to receive the Directive Status Word

**Macro Call**

```
RDEF$ efn
```

efn event flag number

**Macro Expansion**

```
RDEF$ 6
.BYTE 37.,2
.WORD 6
```

**Local Symbol Definitions**

The following symbol is locally defined with its assigned value equal to the byte offset from the start of the DPB to the DPB element:

R.DEEF event flag number (length of 2 bytes)

**DSW Return Codes**

IS.CLR flag was clear

IS.SET flag was set

IE.IEF invalid event flag number (event flag number <1 or >64)

IE.ADP part of DPB is out of issuing task's address space

IE.SDP DIC or DPB size is invalid

## RDXF\$

### 9.1.44 RDXF\$—Read Extended Event Flags

The Read Extended Event Flags directive instructs the system to read all local and common event flags for the issuing task and record their polarity in a 96-bit (6-word) buffer.

#### Fortran Call

A Fortran task can read only one event flag. The call is:

```
CALL READEF (efn[,ids])
```

efn     event flag number

ids     directive status

The Executive returns the status codes IS.SET (+02) and IS.CLR (00) for Fortran calls to report event flag polarity.

#### Macro Call

```
RDXF$ buf
```

#### Buffer Format

Word 0	task Local Flags 1–16
Word 1	task Local Flags 17–32
Word 2	task Common Flags 33–48
Word 3	task Common Flags 49–64
Word 4	reserved for future use
Word 5	reserved for future use

#### Macro Expansion

```
RDXF$     FLGBUF
.BYTE     39.,3             ;RDXF$ MACRO DIC, DPB SIZE=3 WORDS
.WORD     FLGBUF           ;ADDRESS OF 6-WORD BUFFER
```

#### Local Symbol Definitions

R.DABA     buffer address (2)

**DSW Return Codes**

IS.SUC	successful completion
IS.CLR	group global event flags do not exist; words 4 and 5 of the buffer contain 0
IE.ADP	part of the DPB or buffer is out of the issuing task's address space
IE.SDP	DIC or DPB size is invalid

## RPOI\$

### 9.1.45 RPOI\$—Request and Pass Offspring Information

The Request and Pass Offspring Information directive instructs the system to request the specified task, and to chain to it by passing any or all of the parent connections from the issuing task to the requested task. Optionally, the directive can pass a command line to the requested task. Only a privileged task may specify the UIC and TI: of the requested task.

#### Fortran Call

```
CALL RPOI (tname,[iugc],[iumc],[iparen],[ibuf],[ibfl], [isc],
          [idnam],[iunit],[itask],[ocbad][,ids])
```

tname	name of an array containing the actual name (in Radix-50) of the task to be requested and optionally chained to
iugc	name of an integer containing the group code number for the UIC of the requested target chain task
iumc	name of the integer containing the member code number for the UIC of the requested target chain task
iparen	name of an array (or I*4 integer) containing the Radix-50 name of the parent task; this name is returned in the information buffer of the GTCMC! subroutine
ibuf	name of an array that contains the command line text for the chained task
ibfl	name of an integer that contains the number of bytes in the command in the ibuf array
isc	flag byte controlling the actions of this directive request when executed; the bit definitions of this byte (only the low order byte of the integer specified in the call is ever used) are as follows: RP.OEX = 128 force this task to exit on successful execution of the RPOI directive RP.OAL = 1 pass all of this task's connections to the requested task (the default is none) RP.ONX = 2 pass the first connection in the queue if there is one
idnam	name of an integer containing the ASCII device name of the requested task's TI
iunit	name of an integer containing the unit number of the requested task's TI: device
itask	name of an array which contains the Radix-50 name the requested task is to run under  any task may specify a new name for the requested task. However, the requested task (specified in the tname parameter) must be installed in the ...task format.

ocbad reserved for future use  
 ids name of an integer to receive the Directive Status Word

### Macro Call

RPOI\$ tname,,,[ugc],[umc],[parent],[bufadr],[buflen], [sc],  
 [dnam],[unit],[task],[ocbad]

tname name of task to be chained to  
 ugc group code for UIC of the requested task  
 umc member code for UIC of the requested task  
 parent name of issuing task's parent task whose connection is to be passed; if not specified, all connections are passed  
 bufadr address of buffer to be given to the requested task  
 buflen length of buffer to be given to requested task  
 sc flags byte:  
 RP.OEX - (200) force issuing task to exit  
 RP.OAL - (1) pass all connections (default is none)  
 RP.ONX - (2) pass the first connection in the queue, if there is one  
 dnam ASCII device name for TI:  
 unit unit number of task TI:  
 task radix-50 name that the requested task is to run under.  
 Any task may specify a new name for the requested task. However, the requested task (specified in the tname parameter) must be installed in the ...tsk format.  
 ocbad reserved for future use

### Local Symbol Definitions

R.POTK radix-50 name of task to be chained to (2)  
 R.POUM UIC member code (1)  
 R.POUG UIC group code  
 R.POPT name of parent whose OCB should be passed (4)  
 R.POBF address of command buffer (2)  
 R.POBL length of command (2)  
 R.POUN unit number of task TI: (1)  
 R.POSC flags byte (1)  
 R.PODV ASCII device name for TI: (2)  
 R.POTN radix-50 name of task to be started (4)

**Macro Expansion**

```

RPOI$   tname,,,ugc,umc,ptsk,buf,buflen,sc,dev,unit,task,ocbad
.BYTE   11,16           ;DIC 11 DPB SIZE = 16. words
.RAD50  /tname/        ;NAME OF TASK TO CHAIN TO
.BLKW   3              ;RESERVED
.BYTE   umc            ;UIC MEMBER CODE
.BYTE   ugc            ;UIC GROUP CODE
.RAD50  ptsk           ;NAME OF TASK WHOSE OCB SHOULD BE PASSED
.WORD   ocbad          ;ADDRESS OF OCB
.WORD   buf            ;ADDRESS OF BUFFER TO SEND
.WORD   buflen         ;LENGTH OF BUFFER
.ASCII  /dev/          ;ASCII NAME OF TI: OF REQUESTED TASK
.BYTE   unit           ;UNIT NUMBER OF TI: DEVICE
.BYTE   sc             ;PASS BUFFER AS SEND PACKET OR COMMAND
                        ;CODE

```

**DSW Return Codes**

IE.UPN      there is insufficient dynamic memory to allocate an Offspring Control Block, command line buffer, Task Control Block, or Partition Control Block

IE.INS      the specified task was not installed but no command line was specified

IE.ACT      the specified task was already active and it was not a command line interpreter

IE.IDU      the specified virtual terminal unit does not exist or was not created by the issuing task

IE.NVR      there is no Offspring Control Block from the specified parent task

IE.ALG      either a parent name or an offspring control block address was specified and the pass all connections flag or the pass next connection flag was set

IE.PNS      the Task Control Block cannot be created in the same partition as its prototype

IE.ADP      part of the DPB, exit status block, or command line is out of the issuing task's address space

IE.SDP      DIC or DPB size is invalid

**RQST\$****9.1.46 RQST\$—Request Task**

The Request Task directive instructs the system to activate a task. The task is activated and subsequently runs contingent upon priority and memory availability. The Request Task directive is the basic mechanism used by running tasks to initiate other installed (dormant) tasks. The Request Task directive is a frequently used subset of the Run directive. See Notes.

**Fortran Call**

```
CALL REQUES (tsk,[opt][,ids])
```

tsk    task name  
 opt    a 4-word integer array  
       opt(1) partition name first half; ignored, but must be present  
       opt(2) partition name second half; ignored, but must be present  
       opt(3) priority; ignored, but must be present  
       opt(4) user Identification Code  
 ids    directive status

**Macro Call**

```
RQST$ tsk,[prt],[pri][,ugc,umc]
```

tsk    task name  
 prt    partition name; ignored, but must be present  
 pri    priority; ignored, but must be present  
 ugc    UIC group code  
 umc    UIC member code

**Macro Expansion**

```
RQST$    ALPHA , , , 20 , 10
.BYTE    11 . , 7            ;RQST$ MACRO DIC, DPB SIZE=7 WORDS
.RAD50   /ALPHA/            ;TASK ''ALPHA''
.WORD    0 , 0              ;PARTITION IGNORED
.WORD    0                  ;PRIORITY IGNORED
.BYTE    10 , 20            ;UIC UNDER WHICH TO RUN TASK
```

**Local Symbol Definitions**

R.QSTN    task name (4)  
 R.QSPN    partition name (4)

R.QSPR	priority (2)
R.QSGC	UIC group (1)
R.QSPC	UIC member (1)

### DSW Return Codes

IS.SUC	successful completion
IE.UPN	insufficient dynamic memory
IE.INS	task is not installed
IE.ACT	task is already active
IE.ADP	part of the DPB is out of the issuing task's address space
IE.SDP	DIC or DPB size is invalid

### Notes

1. The requested task must be installed in the system.
2. If the partition in which a requested task is to run is already occupied, the Executive places the task in a queue of tasks waiting for that partition. The requested task then runs, depending on priority, and resource availability, when the partition is free.  
  
If the current occupant of the partition is checkpointable, has checkpointing enabled, and is of lower priority than the requested task, it is written to disk when its current outstanding I/O completes; the requested task is then read into the partition.
3. Successful completion means that the task has been declared active, not that the task is actually running.
4. The requested task acquires the same TI: assignment as that of the requesting task.
5. The requested task always runs at the priority specified in its task header.
6. A task that executes in a system-controlled partition requires dynamic memory for the Partition Control Block used to describe its memory requirements.
7. Each active task has two UICs: a protection UIC and a default UIC. These are both returned when a task issues a Get Task Parameters directive (GTSK\$). The UICs are used in the following way:
  - The protection UIC determines the task's access rights for opening files and attaching to regions. When a task attempts to open a file, the system compares the task's protection UIC against the protection mask of the specified UFD; the comparison determines whether the task is to be considered for system, owner, group, or world access.



**RREF\$****9.1.47 RREF\$—Receive By Reference**

The Receive By Reference directive requests the Executive to dequeue the next packet in the receive-by-reference queue of the issuing (receiver) task. Optionally, the task exits if there are no packets in the queue. The directive may also specify that the Executive proceed to map the referred region.

If successful, the directive declares a significant event.

Each reference in the task's receive-by-reference queue represents a separate attachment to a region. If a task has multiple references to a given region, it is attached to that region the corresponding number of times. Because region attachment requires system dynamic memory, the receiver task should detach from any region that it was already attached to in order to prevent depletion of the memory pool. That is, the task needs to be attached to a given region only once.

If the Executive does not find a packet in the queue, and the task has set WS.RCX in the window status word (W.NSTS), the task exits. If WS.RCX is not set, the Executive returns the DSW code IE.ITS.

If the Executive finds a packet, it writes the information provided to the corresponding words in the Window Definition Block. This information provides sufficient information to map the reference, according to the sender task's specifications, to a previously created address window.

If the address of a 10-word receive buffer has been specified (W.NSRB in the Window Definition Block), then the sender task name and the eight additional words passed by the sender task (if any) are placed in the specified buffer. If the sender task did not pass any additional information, the Executive writes the sender task name and eight words of zero.

If the WS.MAP bit in the window status word has been set to 1, the Executive transfers control to the Map Address Window directive to attempt to map the reference.

When a task that has unreceived packets in its receive-by-reference queue exits or is removed, the Executive removes the packets from the queue and deallocates them. Any related flags are not set.

**Fortran Call**

CALL RREF (iwdb,[isrb][,ids])

iwdb an 8-word integer array containing a Window Definition Block (see Section 7.5.2.2)

isrb a 10-word integer array to be used as the receive buffer. If the call omits this parameter, the contents of iwdb(8) are unchanged.

ids directive status

**Macro Call**

```
RREF$ wdb
```

wdb Window Definition Block address

**Macro Expansion**

```
RREF$   WDBADR
.BYTE   81.,2   ;RREF$ MACRO DIC, DPB SIZE=2 WORDS
.WORD   WDBADR   ;WDB ADDRESS
```

Table 9-9  
Window Definition Block Parameters

*Input Parameters*

<i>Array Element</i>	<i>Offset</i>	<i>Description</i>
iwdb(1) bits 0-7	W.NID	ID of an existing window if region is to be mapped
iwdb(7)	W.NSTS	Bit settings <sup>15</sup> in the window status word:
		<i>Bit</i> <i>Definition</i>
		WS.MAP        1 if received reference is to be mapped
		WS.RCX        1 if task exit desired when no packet is found in the queue
iwdb(8)	W.NSRB	Optional address of a 10-word buffer to contain the sender task name and additional information

*Output Parameters*

iwdb(4)	W.NRID	Region ID (pointer to attachment description)
---------	--------	---

15. If you are a Fortran programmer, refer to Section 7.5.2 to determine the bit values represented by the symbolic names described.

**Local Symbol Definitions**

R.REBA        window Definition Block address (2)

**DSW Return Codes**

IS.SUC        successful completion  
IS.HWR        region has incurred a parity error  
IE.ITS        no packet found in the receive-by-reference queue  
IE.ADP        address check of the DPB, WDB, or the receive buffer  
              (W.NSRB) failed  
IE.SDP        DIC or DPB size is invalid

## RSUM\$

### 9.1.48 RSUM\$—Resume Task

The Resume Task directive instructs the system to resume the execution of a task that has issued a Suspend directive.

#### Fortran Call

```
CALL RESUME (tsk[,ids])
```

tsk     task name  
ids     directive status

#### Macro Call

```
RSUM$ tsk
```

tsk     task name

#### Macro Expansion

```
RSUM$     ALPHA  
.BYTE     47.,3             ;RSUM$ MACRO DIC, DPB SIZE=3 WORDS  
.RAD50    /ALPHA/         ;TASK ''ALPHA''
```

#### Local Symbol Definitions

R.SUTN     task name (4)

#### DSW Return Codes

IS.SUC     successful completion  
IE.INS     task is not installed  
IE.ACT     task is not active  
IE.ITS     task is not suspended  
IE.ADP     part of the DPB is out of the issuing task's address space  
IE.SDP     DIC or DPB size is invalid

**RUN\$****9.1.49 RUN\$—Run Task**

The Run Task directive causes a task to be requested at a specified time, and optionally to be requested periodically. The scheduled time is specified in terms of delta time from issuance. If the smg, rmg, and rnt parameters are omitted, Run is the same as Request Task except that:

1. Run causes the task to become active one clock tick after the directive is issued.
2. The system always sets the TI: device for the requested task, to CO:.

See Notes.

**Fortran Call**

CALL RUN (tsk,[opt],[smg],snt,[rmg],[rnt][,ids])

tsk task name

opt a 4-word integer array

- opt(1) partition name first half; ignored, but must be present
- opt(2) partition name second half; ignored, but must be present
- opt(3) priority; ignored, but must be present
- opt(4) user Identification Code

smg schedule delta magnitude

snt schedule delta unit (either 1, 2, 3, or 4)

rmg reschedule interval magnitude

rnt reschedule interval unit

ids directive status

The ISA standard call for initiating a task is also provided:

CALL START (tsk,smg,snt[,ids])

tsk task name

smg schedule delta magnitude

snt schedule delta unit (either 0, 1, 2, 3, or 4)

ids directive status

**Macro Call**

RUN\$ tsk,[prt],[pri],[ugc],[umc],[smg],snt[,rmg,rnt]

tsk task name

prt partition name; ignored, but must be present  
 pri priority; ignored, but must be present  
 ugc UIC group code  
 umc UIC member code  
 smg schedule delta magnitude  
 snt schedule delta unit (either 1, 2, 3, or 4)  
 rmg reschedule interval magnitude  
 rnt reschedule interval unit

### Macro Expansion

```

RUN$      ALPHA,,,20,10,20.,3,10.,3
BYTE     17.,11.      ;RUN$ MACRO DIC, DPB SIZE=11.  WORDS
.RAD50   /ALPHA/      ;TASK ''ALPHA''
.WORD    0,0          ;PARTITION IGNORED
.WORD    0            ;PRIORITY IGNORED
.BYTE    10,20       ;UIC TO RUN TASK UNDER
.WORD    20.         ;SCHEDULE MAGNITUDE=20
.WORD    3           ;SCH. DELTA TIME UNIT=MINUTE (=3)
.WORD    10.        ;RESCH. INTERVAL MAGNITUDE=10.
.WORD    3           ;RESCH. INTERVAL UNIT=MINUTE (=3)

```

### Local Symbol Definitions

R.UNTN task name (4)  
 R.UNPN partition name (4)  
 R.UNPR priority (2)  
 R.UNGC UIC group code (1)  
 R.UNPC UIC member code (1)  
 R.UNSM schedule magnitude (2)  
 R.UNSU schedule unit (2)  
 R.UNRM reschedule magnitude (2)  
 R.UNRU reschedule unit (2)

### DSW Return Codes

For CALL RUN and RUN\$:

IS.SUC successful completion  
 IE.UPN insufficient dynamic memory  
 IE.ACT multiuser task name specified  
 IE.INS task is not installed

IE.PRI	nonprivileged task specified a UIC other than its own.
IE.ITI	invalid time parameter
IE.ADP	part of the DPB is out of the issuing task's address space
IE.SDP	DIC or DPB size is invalid

For CALL START:

The system provides the following positive error codes to be returned for ISA calls:

2	insufficient dynamic storage
3	specified task not installed
94	invalid time parameter
98	invalid event flag number
99	part of DPB is out of task's address space
100	DIC or DPB size is invalid

### Notes

1. A nonprivileged task cannot specify a UIC that is not equal to its own protection UIC. A privileged task can specify any UIC.
2. The target task must be installed in the system.
3. If there is not enough room in the partition in which a requested task is to run, the Executive places the task in a queue of tasks waiting for that partition. The requested task then runs, depending on priority and resource availability, when the partition is free. Another possibility is that checkpointing may occur. If the current occupant of the partition is checkpointable, has checkpointing enabled, is of lower priority than the requested task, or is stopped for terminal input, it is written to disk when its current outstanding I/O completes. The requested task is then read into the partition.
4. Successful completion means the task has been made active; it does not mean that the task is actually running.
5. The Executive returns the code IE.ITI in the DSW if the directive specifies an invalid time parameter. A time parameter consists of the time interval magnitude, and the time interval unit.

A legal magnitude value (smg or rmg) is related to the value assigned to the time interval unit snt or rnt.

The unit values are encoded as follows:

For an ISA Fortran call (CALL START):

- 0 ticks—a tick occurs for each clock interrupt tick rate of  $64_{10}$  ticks per second
- 1 milliseconds—the subroutine converts the specified magnitude to the equivalent number of system clock ticks

For all other Fortran and all macro calls:

- 1 ticks—a tick occurs for each clock interrupt tick rate of  $64_{10}$  ticks per second

For both types of Fortran calls and all macro calls:

- 2 seconds
- 3 minutes
- 4 hours



**SDAT\$****9.1.50 SDAT\$—Send Data**

The Send Data directive instructs the system to declare a significant event and to queue (FIFO) a 13-word block of data for a task to receive.

Note: When a local event flag is specified, the indicated event flag is set for the sending task; a significant event is always declared.

**Fortran Call**

```
CALL SEND (tsk,buf,[efn][,ids])
```

tsk    task name  
buf    13-word integer array of data to be sent  
efn    event flag number  
ids    directive status

**Macro Call**

```
SDAT$ tsk,buf[,efn]
```

tsk    task name  
buf    address of 13-word data buffer  
efn    event flag number

**Macro Expansion**

```
SDAT$    ALPHA , DATBUF , 52 .  
.BYTE    71 . , 5        ;SDAT$ MACRO DIC , DPB SIZE=5 WORDS  
.RAD50   /ALPHA/        ;RECEIVER TASK NAME  
.WORD    DATBUF        ;ADDRESS OF 13.-WORD BUFFER  
.WORD    52 .            ;EVENT FLAG NUMBER 52 .
```

**Local Symbol Definitions**

S.DATN    task name (4)  
S.DABA    buffer address (2)  
S.DAEF    event flag number (2)

**DSW Return Codes**

IS.SUC    successful completion  
IE.INS    receiver task is not installed  
IE.UPN    insufficient dynamic memory

IE.IEF	invalid event flag number (EFN<0 or EFN>64)
IE.ADP	part of the DPB or data block is out of the issuing task's address space
IE.SDP	DIC or DPB size is invalid

**Notes**

1. Send Data requires dynamic memory.
2. If the directive specifies a local event flag, the flag is local to the sender (issuing) task.

Normally, the event flag is used to trigger the receiver task into some action. For this purpose, the event flag must be common (33 through 64) rather than local. (Refer to the descriptions of the Receive Data directive and the Exit IF directive.)

3. The Send Data directive is treated as a 13-word Variable Send Data directive.

**SDIR\$****9.1.51 SDIR\$—Setup Default Directory String**

The Setup Default Directory String directive establishes a single default directory string for the system. (See the Get Default Directory directive.)

**Fortran Call**

```
CALL SETDDS (mod,iens,ienssz,[idsw])
```

mod	the modifier of the logical name within a table
iens	character array containing the equivalence name string
ienssz	size (in bytes) of the equivalence name string
idsw	integer to receive the Directive Status Word

**Macro Call**

```
SDIR$ mod,ens,enssz
```

mod	the modifier of the logical name within a table
ens	buffer address of the equivalence name string
enssz	size (in bytes) of the equivalence name string

**Macro Expansion**

```
SDIR$  mod,ens,enssz
.BYTE  207.,5      ;SDIR$ MACRO DIC, DPB SIZE = 5 WORDS
.BYTE  3           ;SUBFUNCTION CODE FOR SET DEFAULT
                   ;DIRECTORY
.BYTE  MOD        ;LOGICAL NAME MODIFIER
.WORD  0          ;RESERVED
.WORD  ENS        ;BUFFER ADDRESS OF EQUIVALENCE NAME
.WORD  ENSSZ     ;BYTES COUNT OF EQUIVALENCE STRING
```

**Local Symbol Definitions**

S.DENS	address of equivalence name buffer (2)
S.DESZ	byte count of equivalence name string (2)
S.DFUN	subfunction code (1)
S.DMOD	logical name modifier (1)

**DSW Return Codes**

IS.SUC	successful completion of service
IS.SUP	successful completion of service; a new equivalence name string superseded a previously specified name string
IE.UPN	insufficient dynamic storage is available to create the logical name
IE.IBS	the length of the logical or equivalence string is invalid; each string length must be greater than 0 but not greater than 255 <sub>10</sub> characters
IE.ADP	part of the DPB or user buffer is out of the issuing task's address space, or the user does not have proper access to that region
IE.SDP	DIC or DPB size is invalid

**SDRC\$****9.1.52 SDRCS—Send, Request and Connect**

The Send, Request And Connect directive performs a Send Data to the specified receiver task. This action causes the Executive to declare a significant event and to queue (FIFO) a 13-word block of data for the receiver task. The SDRCS directive then Requests the receiver task for execution if the task is not already active. It then Connects to the receiver task (making it an offspring of the sender) by queueing an Offspring Control Block (OCB) to the receiver task's Task Control Block (TCB) and incrementing the rundown count in the sending (parent) task's TCB.

The rundown count is used to inform the Executive that the parent task has one or more offspring tasks; cleanup is necessary if the parent task exits with offspring tasks still active. The rundown count is decremented when the offspring task exits. The OCB contains the TCB address as well as sufficient information to effect all of the specified exit events when the offspring task exits.

If an AST address is specified, an exit AST routine is effected when the offspring task exits with the address of the task's exit status block on the stack. The AST routine must remove this word from the stack before issuing the AST Service Exit directive.

**Fortran Call**

CALL SDRC (rtname, ibuf,[iefn],[iast],[iesb],[iparm][,ids])

rtname	target task name of the offspring task to be connected
ibuf	name of 13-word send buffer
iefn	event flag to be set when the offspring task exits or emits status
iast	name of an AST routine to be called when the offspring task exits or emits status

Note: Refer to Section 3.4.4 for important guidelines on using Fortran AST service routines.

iesb	name of an 8-word status block to be written when the offspring task exits or emits status
	Word 0          offspring task exit status
	Word 1          system abort code
	Word 2-7        reserved

Note: The exit status block defaults to one word. To use the 8-word exit status block, you must specify the logical OR of the symbol SP.WXB and the event flag number in the iefn parameter above.

iparm	name of a word to receive the status block address when an AST occurs
ids	integer to receive the Directive Status Word

**Macro Call**

SDRC\$ tname,buf,[efn],[east],[esb]

tname	target task name of the offspring task to be connected
buf	address of 13-word send buffer
efn	the event flag to be cleared on issuance and set when the offspring task exits or emits status
east	address of an AST routine to be called when the offspring task exits or emits status
esb	address of an 8-word status block to be written when the offspring task exits or emits status
	Word 0          offspring task exit status
	Word 1          system abort code
	Word 2-7        reserved

Note: The exit status block defaults to one word. To use the 8-word exit status block, you must specify the logical OR of the symbol SP.WXB and the event flag number in the efn parameter above.

**Macro Expansion**

```
SDRC$      ALPHA,BUFFR,2,SDRCTR,STBLK
.BYTE      141.,7           ;SDRC$ MACRO DIC, DPB SIZE=7 WORDS
.RAD50     ALPHA           ;TARGET TASK NAME
.WORD      BUFFR          ;SEND BUFFER ADDRESS
.BYTE      2               ;EVENT FLAG NUMBER = 2
.BYTE      16.            ;EXIT STATUS BLOCK CONSTANT
.WORD      SDRCTR         ;ADDRESS OF AST ROUTINE
.WORD      STBLK          ;ADDRESS OF STATUS BLOCK
```

**Local Symbol Definitions**

S.DRTN	task name (4)
S.DRBF	buffer address (2)
S.DREF	event flag (2)
S.DREA	AST routine address (2)
S.DRES	status block address (2)

**DSW Return Codes**

IS.SUC	successful completion
IE.UPN	insufficient dynamic memory to allocate a send packet, Offspring Control Block, Task Control Block, or Partition Control Block

IE.INS	the specified task is an ACP or has the no-send attribute
IE.IEF	an invalid event flag number was specified (EFN<0 or EFN>64)
IE.ADP	part of the DPB or exit status block is not in the issuing task's address space
IE.SDP	DIC or DPB size is invalid

**Notes**

1. The virtual mapping of the exit status block should not be changed while the connection is in effect. Doing so may result in obscure errors.
2. If the directive is rejected, the state of the specified event flag is indeterminate.

## SDRP\$

### 9.1.53 SDRP\$—Send Data Request and Pass Offspring Control Block

This directive instructs the system to send a send data packet for the specified task, chain to the requested task, and request it if it is not already active.

#### Fortran Call

CALL SDRP (task,ibuf,[ibfl],[iefn],[iflag],[iparen],[iocbad] [,ids])

task	name of an array (REAL,INTEGER,I*4) that contains the Radix-50 name of the target task
ibuf	name of an integer array containing the data to be sent
ibfl	name of an integer containing the number of words (integers) in the array to be sent. This argument may be in the range of 1 through 255(10). If this argument is not specified, a default value of 13(10) is assumed.
iefn	name of an integer containing the number of the event flag that is to be set when this directive is executed successfully.
iflag	name of an integer containing the flag bits controlling the execution of this directive. They are defined as follows: SD.REX = 128. force this task to exit upon successful execution of this directive SD.RAL = 1 pass all connections to the requested task (default is pass none); if you specify this flag, do not specify the parent task name SD.RNX = 2 pass the first connection in the queue, if there is one, to the requested task; if you specify this flag, do not specify the parent task name
iparen	name of an array containing the Radix-50 name of the parent task whose connection should be passed to the target task. The name of the parent task was returned in the information buffer of the GTCMCI subroutine.
iocbad	reserved for future use
ids	name of an integer to receive the contents of the Directive Status Word

#### Macro Call

SDRP\$ task,bufadr,[buflen],[efn],[flag],[parent],[ocbad]

task	name of task to be chained to
bufadr	address of buffer to be given to the requested task
buflen	length of buffer to be given to requested task



efn	event flag
flag	flags byte controlling the execution of this directive. The flag bits are defined as follows: SD.REX = 128. force this task to exit upon successful completion of this directive SD.RAL = 1 pass all connections to the requested task (default is pass none); if you specify this flag, do not specify the parent task name SD.RNX = 2 Pass the first connection in the queue, if there is one, to the requested task; if you specify this flag, do not specify the parent task name
parent	name of issuing task's parent task whose connection is to be passed; if not specified, all connections or no connections are passed depending on the flag byte
ocbad	reserved for future use

### Macro Expansion

SDRP\$ task,bufadr,[buflen],[efn],[flag],[parent],[ocbad]

```
.BYTE 141.,9. ;DIC = 141, DPB LENGTH =9 WORDS
.RAD50 /task/ ;TASK NAME IN RADIX-50
.WORD BUFADR ;BUFFER ADDRESS
.BYTE EFN,FLAG ;EVENT FLAG, FLAGS BYTE
.WORD BUFLN ;BUFFER LENGTH
.RAD50 /PARENT/ ;PARENT TASK NAME
.WORD OCBAD ;ADDRESS OF OCB
```

### Local Symbol Definitions

S.DRTK	radix-50 name of task to be chained to
S.DRAD	send data buffer address
S.DREF	event flag
S.DRFL	flags byte: SD.REX - (200) force task to exit (task issuing directive) SD.RAL - (1) pass all connections to the requested task (default is pass none); if you specify this flag, do not specify the parent task name SD.RNX - (2) pass the first connection in the queue, if there is one, to the requested task; if you specify this flag, do not specify the parent task name
S.DRBL	length of send data packet (up through 255 <sub>10</sub> words)
S.DRPT	name of parent whose OCB should be passed
S.DROA	reserved for future use.

**DSW Return Codes**

IE.NVR	no Offspring Control Block from specified parent
IE.ALG	either a parent name or an OCB address was specified and the pass all connections flag was set
IE.IBS	length of send packet is illegal. The send packet may be up through 255 <sub>10</sub> bytes long
IE.UPN	insufficient dynamic memory to allocate a send packet, Offspring Control Block, Task Control Block, or partition control block
IE.INS	the specified task is an ACP or has the no-send attribute
IE.IEF	an invalid event flag number was specified (EFN <0 or EFN >64)
IE.ADP	part of the DPB or exit status block is out of the issuing task's address space
IE.SDP	DIC or DPB size is invalid

**Note**

1. If the directive is rejected, the state of the specified event flag is indeterminate.

**SETF\$****9.1.54 SETF\$—Set Event Flag**

The Set Event Flag directive instructs the system to set an indicated event flag that reports the flag's polarity before setting.

**Fortran Call**

```
CALL SETEF (efn[,ids])
```

efn event flag number

ids directive status

**Macro Call**

```
SETF$ efn
```

efn event flag number

**Macro Expansion**

```
SETF$      52.
.BYTE      33.,2      ;SETF$ MACRO DIC, DPB SIZE=2 WORDS
.WORD      52.      ;EVENT FLAG NUMBER 52.
```

**Local Symbol Definitions**

S.ETEF event flag number (2)

**DSW Return Codes**

IS.CLR flag was clear

IS.SET flag was already set

IE.IEF invalid event flag number (EFN<1 or EFN>64)

IE.ADP part of the DPB is out of the issuing task's address space

IE.SDP DIC or DPB size is invalid

**Note**

1. Set Event Flag does not declare a significant event; it merely sets the specified flag.

## SFPA\$

### 9.1.55 SFPA\$—Specify Floating Point Processor Exception AST

The Specify Floating Point Processor Exception AST directive instructs the system to record one of the two following cases:

- Floating Point Processor exception ASTs for the issuing task are desired, and the Executive is to transfer control to a specified address when such an AST occurs for the task.
- Floating Point Processor exception ASTs for the issuing task are no longer desired.

When an AST service routine entry point address is specified, subsequent Floating Point Processor exception ASTs occur for the issuing task, and control will be transferred to the indicated location at the time of the AST's occurrence. When an AST service entry point address is not specified, subsequent Floating Point Processor exception ASTs do not occur until the task issues a directive that specifies an AST entry point. See Notes.

#### Fortran Call

Not supported

#### Macro Call

```
SFPA$ [ast]
```

ast     AST service routine entry point address

#### Macro Expansion

```
SFPA$     FLTAST
.BYTE     111.,2             ;SFPA$ MACRO DIC, DPB SIZE=2 WORDS
.WORD     FLTAST            ;ADDRESS OF FLOATING POINT AST
```

#### Local Symbol Definitions

S.FPAE     AST entry address (2)

#### DSW Return Codes

IS.SUC	successful completion
IE.UPN	insufficient dynamic memory
IE.ITS	AST entry point address is already unspecified or task was built without floating-point support (FP switch not specified in Task Builder .TSK file specification)
IE.AST	directive was issued from an AST service routine, or ASTs are disabled

IE.ADP	part of the DPB is out of the issuing task's address space
IE.SDP	DIC or DPB size is invalid

**Notes**

1. A Specify Floating Point Processor Exception AST requires dynamic memory.
2. The Executive queues Floating Point Processor exception ASTs when a Floating Point Processor exception trap occurs for the task. No subsequent ASTs of this kind can be queued for the task until the first one queued has actually been effected.
3. The Floating Point Processor exception AST service routine is entered with the task stack in the following state:

SP+12	event flag mask word
SP+10	PS of task prior to AST
SP+06	PC of task prior to AST
SP+04	DSW of task prior to AST
SP+02	floating exception code
SP+00	floating exception address

The task must remove the floating exception code and address from the task's stack before an AST Service Exit directive is executed.

4. This directive cannot be issued either from an AST service routine or when ASTs are disabled.
5. This directive applies only to the Floating Point Processor.

## SPND\$\$

### 9.1.56 SPND\$\$—Suspend (\$S Form Recommended)

The Suspend directive instructs the system to suspend the execution of the issuing task. A task can suspend only itself, not another task. The task can be restarted by a Resume directive.

#### Fortran Call

```
CALL SUSPND [(ids)]
```

ids     directive status

#### Macro Call

```
SPND$$ [err]
```

err     error routine address

#### Macro Expansion

```
SPND$$ ERR
MOV      (PC)+, -(SP)      ;PUSH DPB ONTO THE STACK
.BYTE   45., 1             ;SPND$$ MACRO DIC, DPB SIZE=1 WORD
EMT      377               ;TRAP TO THE EXECUTIVE
BCC      .+6               ;BRANCH IF DIRECTIVE SUCCESSFUL
JSR      PC,ERR            ;OTHERWISE, CALL ROUTINE ''ERR''
```

#### Local Symbol Definitions

None

#### DSW Return Codes

IS.SPD     successful completion (task was suspended)  
 IE.ADP     part of the DPB is out of the issuing task's address space  
 IE.SDP     DIC or DPB size is invalid

#### Notes

1. A suspended task retains control of the system resources allocated to it. The Executive makes no attempt to free these resources until a task exits.
2. A suspended task is eligible for checkpointing unless it is fixed or declared to be noncheckpointable.
3. Because this directive requires only a 1-word DPB, the \$\$ form of the macro is recommended; it requires less space and executes with the same speed as that of the DIR\$ macro.

**SPWN\$****9.1.57 SPWN\$—Spawn**

The Spawn directive requests a specified task for execution, optionally queuing a command line and establishing the task's Tl: as a physical terminal.

When this directive is issued, an Offspring Control Block (OCB) is queued to the offspring TCB and a rundown count is incremented in the parent task's TCB. The rundown count is used to inform the Executive that the task is a parent task and has one or more offspring tasks; cleanup is necessary if a parent task exits with active offspring tasks. The rundown count is decremented when the spawned task exits. The OCB contains the TCB address as well as sufficient information to effect all of the specified exit events when the offspring task exits.

If a command line is specified, it is buffered in the Executive pool and queued for the offspring task for subsequent retrieval by the offspring task with the Get Command Line directive. Maximum command line length is 255<sub>10</sub> characters.

If an AST address is specified, an exit AST routine is effected when the spawned task exits with the address of the task's exit status block on the stack. The AST routine must remove this word from the stack before issuing the AST Service Exit directive.

**Fortran Call**

```
CALL SPAWN (rname,[iugc],[iumc],[iefn],[iast],[iesb],
            [iparm],[icmlin],[icmlen],[iunit],[dnam],[ids])
```

rname	name (RAD50) of the offspring task to be spawned
iugc	group code number for the UIC of the offspring task
iumc	member code number for the UIC of the offspring task
iefn	event flag to be set when the offspring task exits or emits status
iast	name of an AST routine to be called when the offspring task exits or emits status

Note: Refer to Section 3.4.4 for important guidelines on using Fortran AST service routines.

iesb	name of an 8-word status block to be written when the offspring task exits or emits status
	Word 0      offspring task exit status
	Word 1      system abort code
	Word 2-7    reserved

Note: The exit status block defaults to one word. To use the 8-word exit status block, you must specify the logical OR of the symbol SP.WX8 and the event flag number in the iefn parameter above

iparm	name of a word to receive the status block address when the AST occurs
icmlin	name of a command line to be queued for the offspring task
icmlen	length of the command line (255 <sub>10</sub> characters maximum)
iunit	unit number of terminal to be used as the TI: for the offspring task; if a value of 0 is specified, the TI: of the issuing task is propagated; a task must be privileged in order to specify a TI: other than the parent task's TI:
dnam	device name mnemonic
ids	integer to receive the Directive Status Word

**Macro Call**

SPWN\$ tname,,,[ugc],[umc],[efn],[east],[esb],[cmdlin], [cmdlen],  
[unum],[dnam]

tname	name (RAD50) of the offspring task to be spawned
ugc	group code number for the UIC of the offspring task
umc	member code number for the UIC of the offspring task
efn	the event flag to be cleared on issuance and set when the offspring task exits or emits status
east	address of an AST routine to be called when the offspring task exits or emits status
esb	address of an 8-word status block to be written when the offspring task exits or emits status
	Word 0          offspring task exit status
	Word 1          system abort code
	Word 2-7        reserved

Note: The exit status block defaults to one word. To use the 8-word exit status block, you must specify the logical OR of the symbol SP.WX8 and the event flag number in the efn parameter above.

cmdlin	address of a command line to be queued for the offspring task
cmdlen	length of the command line (maximum length is 255 <sub>10</sub> )
unum	unit number of terminal to be used as the TI: for the offspring task; if a value of 0 is specified, the TI: of the issuing task is propagated; a task must be privileged in order to specify a TI: other than the parent task's TI:
dnam	device name mnemonic; if not specified, the default is TI:



**Macro Expansion**

```

SPWN$   ALPHA,,,3,7,1,ASTRUT,STBLK,CMDLIN,72.,0
.BYTE   11.,13.           ;SPWN$ MACRO DIC, DPB SIZE=13 WORDS
.RAD50  ALPHA             ;NAME OF TASK TO BE SPAWNED
.BLKW   3                 ;RESERVED
.BYTE   7,3              ;UMC = 7   UGC = 3
.BYTE   1                 ;EVENT FLAG NUMBER = 1
.BYTE   16.              ;EXIT STATUS BLOCK CONSTANT
.WORD   ASTRUT           ;AST ROUTINE ADDRESS
.WORD   STBLK            ;EXIT STATUS BLOCK ADDRESS
.WORD   CMDLIN           ;ADDRESS OF COMMAND LINE
.WORD   72.              ;COMMAND LINE LENGTH = 72. CHARACTERS
.WORD   2                 ;TERMINAL UNIT NUMBER =0

```

Note: One additional parameter (device name) can be added for a hardware terminal name. For example, TT0: would have the same macro expansion shown above, plus the following:

```
.ASCII /TT/ ;ASCII DEVICE NAME
```

The DPB size will then be 14 words.

**Local Symbol Definitions**

```

S.PWTN   task name (4)
S.PWXX   reserved (6)
S.PWUM   user member code (1)
S.PWUG   user group code (1)
S.PWEF   event flag number (2)
S.PWEA   exit AST routine address (2)
S.PWES   exit status block address (2)
S.PWCA   command line address (2)
S.PWCL   command line length (2)
S.PWVT   terminal unit number (2)
S.PWDN   device name (2)

```

**DSW Return Codes**

```

IS.SUC   successful completion
IE.UPN   insufficient dynamic memory to allocate an Offspring Control
          Block, command line buffer, Task Control Block, or Partition
          Control Block
IE.INS   the specified task was not installed
IE.ACT   the specified task was already active

```

IE.PRI	nonprivileged task attempted to specify an offspring task's TI: to be different from its own
IE.IDU	the specified terminal unit does not exist or the specified TI: device is not a terminal
IE.IEF	invalid event flag number (EFN<0 or EFN>64)
IE.ADP	part of the DPB, exit status block, or command line is out of the issuing task's address space
IE.SDP	DIC or DPB size is invalid

### Notes

1. If the UIC is defaulted, that task is requested to run under the UIC of the parent task. See the notes for the Request Task (RQST\$) directive for more information about task UICs.
2. The virtual mapping of the exit status block should not be changed while the connection is in effect. Doing so may cause obscure errors.
3. The types of operations that a Fortran AST routine may perform are extremely limited. Please refer to Chapter 3 for a list of restrictions.

**SRDA\$****9.1.58 SRDA\$—Specify Receive Data AST**

The Specify Receive Data AST directive instructs the system to record one of the following two cases:

- Receive data ASTs for the issuing task are desired, and the Executive transfers control to a specified address when data has been placed in the task's receive queue
- Receive data ASTs for the issuing task are no longer desired.

When the directive specifies an AST service routine entry point, receive data ASTs for the task subsequently occur whenever data has been placed in the task's receive queue; the Executive transfers control to the specified address.

When the directive omits an entry point address, the Executive disables receive data ASTs for the issuing task. Receive data ASTs do not occur until the task issues another Specify Receive Data AST directive that specifies an entry point address. (See Notes.)

**Fortran Call**

Neither the Fortran language nor the ISA standard permits direct linking to system trapping mechanisms; therefore, this directive is not available to Fortran tasks.

**Macro Call**

```
SRDA$ [ast]
```

ast    AST service routine entry point address

**Macro Expansion**

```
SRDA$    RECAST
.BYTE    107.,2                    ;SRDA$ MACRO DIC, DPB SIZE=2 WORDS
.WORD    RECAST                   ;ADDRESS OF RECEIVE AST
```

**Local Symbol Definitions**

S.RDAE    AST entry address (2)

**DSW Return Codes**

IS.SUC    successful completion  
 IE.UPN    insufficient dynamic memory  
 IE.ITS    AST entry point address is already unspecified

IE.AST	directive was issued from an AST service routine, or ASTs are disabled
IE.ADP	part of the DPB is out of the issuing task's address space
IE.SDP	DIC or DPB size is invalid

**Notes**

1. A Specify Receive Data AST requires dynamic memory.
2. The Executive queues receive data ASTs when a message is sent to the task. No future receive data ASTs will be queued for the task until the first one queued has been effected.
3. The task enters the receive data AST service routine with the task stack in the following state:

SP+06	event flag mask word
SP+04	PS of task prior to AST
SP+02	PC of task prior to AST
SP+00	DSW of task prior to AST

No trap-dependent parameters accompany a receive data AST; therefore, the AST Service Exit directive must be executed with the stack in the same state as when the AST was effected.

4. This directive cannot be issued either from an AST service routine or when ASTs are disabled.

**SREX\$****9.1.59 SREX\$—Specify Requested Exit AST Directive**

The Specify Requested Exit AST directive allows the task issuing the directive to specify the AST service routine to be entered if an attempt is made to abort the task by a directive. This directive allows a task to enter a routine for clean-up instead of abruptly aborting.

If an AST address is not specified, any previously specified exit AST is canceled.

Privileged tasks enter the specified AST routine each time an abort is issued. However, subsequent exit ASTs will not be queued until the first exit AST has occurred.

Nonprivileged tasks enter the specified AST routine only once. Subsequent attempts to abort the task will actually abort the task.

**Fortran Call**

```
CALL SREX (ast,ipblk,ipblk1,[dummy][,ids])
```

ast            name of the externally declared AST subroutine

Note: Refer to Section 3.4.4 for important guidelines on using Fortran AST service routines.

ipblk          name of an integer array to receive the trap-dependent parameters

ipblk1        number of parameters to be returned into the ipblk array

dummy         reserved for future use

ids            name of an optional integer to receive the Directive Status Word

**Macro Call**

```
SREX$ [ast][,dummy]
```

ast            AST service routine entry point address

dummy         reserved for future expansion

**Macro Expansion**

```
SREX$    REQAST
.BYTE    167.,3            ;SREX$ MACRO DIC, DPB SIZE=3 WORDS
.WORD    REQAST           ;EXIT AST ROUTINE ADDRESS
.WORD    0                ;RESERVED FOR FUTURE EXPANSION
```

Note: The DPB length for the SREX\$ form of the directive is three words.

**Local Symbol Definitions**

S.REAE        exit AST routine address (2)

**DSW Return Codes**

IS.SUC        successful completion  
 IE.UPN        insufficient dynamic storage  
 IE.AST        directive was issued from an AST service routine, or ASTs are disabled  
 IE.ITS        ASTs already not desired, or nonprivileged task attempted to respecify or cancel the AST after one had already occurred  
 IE.ADP        Part of the DPB is out of the issuing task's address space  
 IE.SDP        DIC or DPB size is invalid

**Notes**

1. The issuing task can use the information returned on the stack for this directive to decide how to handle the abort attempt.

After specifying a requested exit AST using the SREX\$ form of the directive, the issuing task will enter the AST service routine if any attempt is made to abort the task. Nonprivileged abort attempts must originate from the same TI: as that of the issuing task.

When the AST service routine is entered and the AST has been specified using the SREX\$ directive, the task's stack is in the following state:

SP+12	event flag mask word
SP+10	PS of task prior to AST
SP+06	PC of task prior to AST
SP+04	DSW of task prior to AST
SP+02	trap-dependent parameter
SP+00	number of bytes to add to SP to clean stack (4)

The trap-dependent parameter is formatted as follows:

Bit 0	0 if the abort attempt was privileged 1 if the abort attempt was nonprivileged
Bit 1	0 if the ABRT\$ directive was issued

Bits 1 through 15 are reserved for future use

The task must remove the trap-dependent parameters from the stack before an AST Service Exit directive is executed. The recommended method is to add the value stored in SP+00 to SP. This is also the only recommended way to access the non-trap-dependent parameters on the stack.

2. The event flag mask word at the bottom of the stack preserves the Wait For conditions of a task prior to AST entry. A task can, after an AST, return to a Wait For state. Because these flags and other stack data are in the user task, they can be modified. However, modifying the stack data may cause unpredictable results. Therefore, such modification is not recommended.
3. Please see Chapter 3 for a list of restrictions on operations that can be performed in a Fortran AST routine.

## SREF\$

### 9.1.60 SREF\$—Send By Reference

The Send By Reference directive inserts a packet containing a reference to a region into the receive-by-reference queue of a specified (receiver) task. The Executive automatically attaches the receiver task for each Send By Reference directive issued by the task to the specified region (the region identified in W.NRID of the Window Definition Block). The attachment occurs even if the receiver task is already attached to the region, unless bit WS.NAT in W.NSTS of the Window Definition Block is set. The successful execution of this directive causes a significant event to occur.

The send packet contains:

- A pointer to the created attachment descriptor, which becomes the region ID to be used by the receiver task
- The offset and length words specified in W.NOFF and W.NLEN of the Window Definition Block (which the Executive passes without checking)
- The receiver task's permitted access to the region, contained in the window status word W.NSTS
- The sender task name
- Optionally, the address of an 8-word buffer that contains additional information (If the packet does not include a buffer address, the Executive sends 8 words of 0.)

The receiver task automatically has access to the entire region as specified in W.NSTS. The sender task must be attached to the region with at least the same types of access. By setting all the bits in W.NSTS to 0, the receiver task can default the permitted access to that of the sender task.

If the directive specifies an event flag, the Executive sets the flag in the sender task (when the receiver task acknowledges the reference) by issuing the Receive By Reference directive. When the sender task exits, the system searches for any unreceived references that specify event flags, and prevents any invalid attempts to set the flags. The references themselves remain in the receiver task's receive-by-reference queues.

#### Fortran Call

```
CALL SREF (tsk,[efn],iwdb,[isrb][,ids])
```

- |      |  |
|------|--|
| tsk  | a single-precision, floating-point variable containing the name of the receiving task in Radix-50 format |
| efn  | event flag number  |
| iwdb | an 8-word integer array containing a Window Definition Block (see Section 7.5.2.2)                       |



- isrb an 8-word integer array containing additional information (If specified, the address of isrb is placed in iwdb(8); if isrb is omitted, the contents of iwdb(8) remain unchanged)
- ids directive status

**Macro Call**

SREF\$ task,wdb[,efn]

- task name of the receiver task
- wdb Window Definition Block address
- efn event flag number

**Macro Expansion**

```
SREF$ ALPHA ,WDBADR , 48 .
.BYTE 69 . , 5 ;SREF$ MACRO DIC , DPB SIZE=5 WORDS
.RAD50 /ALPHA/ ;RECEIVER TASK NAME
.WORD 48 . ;EVENT FLAG NUMBER
.WORD WDBADR ;WDB ADDRESS
```

Table 9-10  
Window Definition Block Parameters

*Input Parameters*

<i>Array Element</i>	<i>Offset</i>	<i>Description</i>										
iwdb(4)	W.NRID	ID of the region to be sent by reference										
iwdb(5)	W.NOFF	Offset word, passed without checking										
iwdb(6)	W.NLEN	Length word, passed without checking										
iwdb(7)	W.NSTS	Bit settings <sup>16</sup> in window status word (the receiver task's permitted access):										
		<table border="0"> <thead> <tr> <th><i>Bit</i></th> <th><i>Definition</i></th> </tr> </thead> <tbody> <tr> <td>WS.RED</td> <td>1 if read access is permitted</td> </tr> <tr> <td>WS.WRT</td> <td>1 if write access is permitted</td> </tr> <tr> <td>WS.EXT</td> <td>1 if extend access is permitted</td> </tr> <tr> <td>WS.DEL</td> <td>1 if delete access is permitted</td> </tr> </tbody> </table>	<i>Bit</i>	<i>Definition</i>	WS.RED	1 if read access is permitted	WS.WRT	1 if write access is permitted	WS.EXT	1 if extend access is permitted	WS.DEL	1 if delete access is permitted
<i>Bit</i>	<i>Definition</i>											
WS.RED	1 if read access is permitted											
WS.WRT	1 if write access is permitted											
WS.EXT	1 if extend access is permitted											
WS.DEL	1 if delete access is permitted											
iwdb(8)	W.NSRB	Optional address of an 8-word buffer containing additional information										

*Output Parameters*

None

16. If you are a Fortran programmer, refer to Section 7.5.2 to determine the bit values represented by the symbolic names described.

**Local Symbol Definitions**

S.RETN	receiver task name (4)
S.REBA	Window Definition Block base address (2)
S.REEF	event flag number (2)

**DSW Return Codes**

IS.SUC	successful completion
IE.UPN	a send packet or an attachment descriptor could not be allocated
IE.INS	the sender task attempted to send a reference to an Ancillary Control Processor (ACP) task, or task not installed
IE.PRI	specified access not allowed to sender task itself
IE.NVR	invalid region ID
IE.IEF	invalid event flag number (EFN<0 or EFN>64)
IE.HWR	region had load failure or parity error
IE.ADP	the address check of the DPB, the WDB, or the send buffer failed
IE.SDP	DIC or DPB size is invalid

**Notes**

1. For the user's convenience, the ordering of the SREF\$ macro arguments does not directly correspond to the format of the DPB. The arguments have been arranged so that the optional argument (efn) is at the end of the macro call. This arrangement is also compatible with the SDAT\$ macro.
2. Because region attachment requires system dynamic memory, the receiver task should detach from any region to which it was already attached, in order to prevent depletion of the memory pool; that is, the task needs to be attached to a given region only once.

**SRRAS****9.1.61 SRRAS—Specify Receive-by-Reference AST**

The Specify Receive-By-Reference AST directive instructs the system to record one of the following two cases:

- Receive-by-reference ASTs for the issuing task are desired, and the Executive transfers control to a specified address when such an AST occurs.
- Receive-by-reference ASTs for the issuing task are no longer desired.

When the directive specifies an AST service routine entry point, receive-by-reference ASTs for the task will occur. The Executive will transfer control to the specified address.

When the directive omits an entry point address, the Executive prevents the occurrence of receive-by-reference ASTs for the issuing task. Receive-by-reference ASTs will not occur until the task issues another Specify Receive-By-Reference AST directive that specifies an entry point address. See Notes.

**Fortran Call**

Neither the Fortran language nor the ISA standard permits direct linking to system trapping mechanisms; therefore, this directive is not available to Fortran tasks.

**Macro Call**

SRRAS [ast]

ast     AST service routine entry point address (0)

**Macro Expansion**

```

SRRAS    RECAST
.BYTE    21.,2                   ;SRRAS MACRO DIC, DPB SIZE=2 WORDS
.WORD    RECAST                 ;ADDRESS OF RECEIVE AST

```

**Local Symbol Definitions**

S.RRAEAST entry address (2)

**DSW Return Codes**

IS.SUC	successful completion
IE.UPN	insufficient dynamic memory.
IE.ITS	AST entry point address is already unspecified

IE.AST	directive was issued from an AST service routine, or ASTs are disabled
IE.ADP	part of the DPB is out of the issuing task's address space
IE.SDP	DIC or DPB size is invalid

**Notes**

1. Specify Receive-By-Reference AST requires dynamic memory.
2. The Executive queues receive-by-reference ASTs when a message is sent to the task. Future receive-by-reference ASTs will not be queued for the task until the first one queued has been effected.
3. The task enters the receive-by-reference AST service routine with the task stack in the following state:

SP+06	event flag mask word
SP+04	PS of task prior to AST
SP+02	PC of task prior to AST
SP+00	DSW of task prior to AST

No trap-dependent parameters accompany a receive-by-reference AST; therefore, the AST Service Exit directive must be executed with the stack in the same state as when the AST was effected.

4. This directive cannot be issued either from an AST service routine or when ASTs are disabled.

**STIM\$****9.1.62 STIM\$—Set System Time**

The Set System Time directive instructs the system to set the system's internal time to the specified time parameters. Optionally, the Set System Time directive returns the system's current internal time to the issuing task before setting the system time to the specified values.

All time parameters must be specified as binary numbers.

A task must be privileged to issue this directive.

When this directive changes the system time by a specified amount, it also effectively changes the time of anything resident on the clock queue by the same amount. Thus, the time synchronization of events is maintained.

**Fortran Call**

```
CALL SETTIM (ibufn[,ibufp[,ids]])
```

ibufn an 8-word integer array (new time specification buffer)

ibufp an 8-word integer array—previous time buffer

ids directive status

**Macro Call**

```
STIM$ bufn,[bufp]
```

bufn address of 8-word new time specification buffer

bufp address of 8-word buffer to receive the previous system time parameters

**Buffer Format**

Word 0 year (since 1900)

Word 1 month (1-12)

Word 2 day (1-n, where n is the highest day possible for the given month and year)

Word 3 hour (0-23)

Word 4 minute (0-59)

Word 5 second (0-59)

Word 6 tick of second (0-n, where n is the frequency of the system clock minus one); if the next parameter (ticks per second) is defaulted, this parameter is ignored

Word 7            ticks per second (must be defaulted or must match the frequency of the system clock at 64. ticks per second); this parameter is used to verify the intended granularity of the "tick of second" parameter

Note: If any of the specified new time parameters are defaulted (equal to -1), the corresponding previous system time parameters will remain unchanged and will be substituted for the defaulted parameters during argument validation.

### Macro Expansion

```

STIM$   NEWTIM,OLDTIM
.BYTE   61.,3           ;STIM$ DIC, DPB SIZE=3 WORDS
.WORD   NEWTIM         ;ADDRESS OF 8.-WORD INPUT BUFFER
.WORD   OLDTIM        ;ADDRESS OF 8.-WORD OUTPUT BUFFER

```

### Local Symbol Definitions

S.TIBA        input buffer address (2)  
S.TIBO        output buffer address (2)

The following offsets are assigned relative to the start of each time parameters buffer:

S.TIYR        year (2)  
S.TIMO        month (2)  
S.TIDA        day (2)  
S.TIHR        hour (2)  
S.TIMI        minute (2)  
S.TICS        second (2)  
S.TICT        clock tick of second (2)  
S.TICP        clock ticks per second (2)

### DSW Return codes

IS.SUC        successful completion  
IE.PRI        the issuing task is not privileged  
IE.ITI        one of the specified time parameters is out of range, or both the tick-of-second parameter and the ticks-per-second parameter were specified and the ticks-per-second parameter does not match the system's clock frequency; the system time at the moment the directive is issued (returned in the second buffer) can be useful in determining the cause of the fault if any of the specified time parameters were defaulted

IE.ADP	part of the DPB or one of the buffers is out of the issuing task's address space
IE.SDP	DIC or DPB size is invalid

**Notes**

1. The buffers used in this directive are compatible with those of the Get Time Parameters (GTIM\$) directive.
2. The second buffer (previous time) is only filled in if the directive was successfully executed or if it was rejected with an error code of IE.ITI.

## STLO\$

### 9.1.63 STLO\$—Stop For Logical OR Of Event Flags

The Stop For Logical OR Of Event Flags directive instructs the system to stop the issuing task until the Executive sets one or more of the indicated event flags from one of the following groups:

- GR 0 local flags 1–16
- GR 1 local flags 17–32
- GR 2 common flags 33–48
- GR 3 common flags 49–64

The task does not stop itself if any of the indicated flags are already set when the task issues the directive. This directive cannot be issued at AST state.

A task that is stopped for one or more event flags can only become unstopped by setting the specified event flag; it cannot become unstopped with the Unstop directive.

#### Fortran Call

```
CALL STLOR (ief1,ief2,ief3, ... ief(n))
```

ief1 ... ief(n) list of event flag numbers

#### Macro Call

```
STLO$ grp, msk
```

grp desired group of event flags

msk a 16-bit mask word

#### Macro Expansion

```
STLO$ 1,47
.BYTE 137.,3 ;STLO$ MACRO DIC, DPB SIZE=3 WORDS
.WORD 1 ;GROUP 1 FLAGS (FLAGS 17-32)
.WORD 47 ;MASK WORD = 47 (FLAGS 17, 18, 19, 22)
```

#### Local Symbol Definitions

S.TLGR group flags (2)

S.TLMS mask word (2)



**DSW Return Codes**

IS.SUC	successful completion
IE.AST	the issuing task is at AST state
IE.IEF	an event flag group other than 0 through 3 was specified, or the event flag mask word is zero
IE.ADP	part of the DPB is out of the issuing task's address space
IE.SDP	DIC or DPB size is invalid

**Notes**

1. There is a one-to-one correspondence between bits in the mask word and the event flags in the specified group; that is, if group 1 were specified (as in the above macro expansion example), bit 0 in the mask word would correspond to event flag 17, bit 1 to event flag 18, and so forth.
2. The Executive does not arbitrarily clear event flags when Stop For Logical OR Of Event Flags conditions are met. Some directives (Queue I/O Request, for example) implicitly clear a flag; otherwise, they must be explicitly cleared by a Clear Event Flag directive.
3. The argument list specified in the Fortran call must contain only event flag numbers that lie within one event flag group. If event flag numbers are specified that lie in more than one group, or if an invalid event flag number is specified, a fatal Fortran error is generated.
4. Tasks stopped for event flag conditions cannot be unstopped by issuing the Unstop directive; tasks stopped in this manner can only be unstopped by meeting event flag conditions.
5. The grp operand must always be of the form n regardless of the macro form used. In all other macro calls, numeric or address values for \$S form macros have the form:  
#n  
For STLO\$\$ this form of the grp argument would be:  
n

## STOP\$\$

### 9.1.64 STOP\$\$—Stop (\$S Form Recommended)

The Stop directive stops the issuing task. This directive cannot be issued at AST state. A task stopped in this manner can only be unstopped by another task that issues an Unstop directive directed to the task or the task issuing an Unstop directive at AST state.

#### Fortran Call

```
CALL STOP ([ids])
```

ids integer to receive the Directive Status Word

#### Macro Call

```
STOP$$
```

#### Macro Expansion

```
STOP$$
MOV      (PC)+, -(SP)      ;PUSH DPB ONTO THE STACK
.BYTE    131., 1          ;STOP$ MACRO DIC, DPB SIZE=1 WORD
EMT      377              ;TRAP TO THE EXECUTIVE
```

#### Local Symbol Definitions

None

#### DSW Return Codes

IS.SET	successful completion
IE.AST	the issuing task is at AST state
IE.ADP	part of the DPB is out of the issuing task's address space
IE.SDP	DIC or DPB size is invalid

**STSE\$****9.1.65 STSE\$—Stop For Single Event Flag**

The Stop For Single Event Flag directive instructs the system to stop the issuing task until the specified event flag is set. If the flag is set at issuance, the task is not stopped. This directive cannot be issued at the AST state.

A task that is stopped for one or more event flags can only become unstopped by setting the specified event flag. The Unstop directive cannot be used to unstop the task.

**Fortran Call**

```
CALL STOPFR (iefn[,ids])
```

iefn event flag number

ids integer to receive Directive Status Word

**Macro Call**

```
STSE$ efn
```

efn event flag number

**Macro Expansion**

```
STSE$ 7
.BYTE 135.,2 ;STSE$ MACRO DIC, DPB SIZE=2 WORDS
.WORD 7 ;LOCAL EVENT FLAG NUMBER = 7
```

**Local Symbol Definitions**

S.TSE event flag number (2)

**DSW Return Codes**

IS.SUC	successful completion
IE.AST	the issuing task is at AST state
IE.IEF	invalid event flag number (EFN<1 or EFN>64)
IE.ADP	part of the DPB is out of the issuing task's address space
IE.SDP	DIC or DPB size is invalid

**Note**

None

## SVDB\$

### 9.1.66 SVDB\$—Specify SST Vector Table For Debugging Aid

The Specify SST Vector Table For Debugging Aid directive instructs the system to record the address of a table of SST service routine entry points for use by an intratask debugging aid (ODT, for example).

To deassign the vector table, omit the parameters `adr` and `len` from the macro call.

Whenever an SST service routine entry is specified in both the table used by the task and the table used by a debugging aid, the trap occurs for the debugging aid, not for the task.

#### Fortran Call

Neither the Fortran language nor the ISA standard permits direct linking to system trapping mechanisms; therefore, this directive is not available to Fortran tasks.

#### Macro Call

```
SVDB$ [adr][,len]
```

`adr` address of SST vector table

`len` length of (number of entries in) the table in words

The vector table has the following format:

Word 0	odd address of nonexistent memory error
Word 1	memory protect violation
Word 2	T-bit trap or execution of a BPT instruction
Word 3	execution of an IOT instruction
Word 4	execution of a reserved instruction
Word 5	execution of a non-RSX EMT instruction (see Note)
Word 6	execution of a TRAP instruction
Word 7	reserved for future use

A 0 entry in the table indicates that the task does not want to process the corresponding SST.

#### Macro Expansion

```
SVDB$ SSTTBL,4
.BYTE 103.,3 ;SVDB$ MACRO DIC, DPB SIZE=3 WORDS
.WORD SSTTBL ;ADDRESS OF SST TABLE
.WORD 4 ;SST TABLE LENGTH=4 WORDS
```

**Local Symbol Definitions**

S.VDTA        table address (2)  
S.VDTL        table length (2)

**DSW Return Codes**

IS.SUC        successful completion  
IE.ADP        part of the DPB or table is out of the issuing task's address  
                 space  
IE.SDP        DIC or DPB size is invalid

**Note**

1. A non-RSX EMT instruction is any EMT instruction not normally used by the system (EMT 1 through 375).

## SVTK\$

### 9.1.67 SVTK\$—Specify SST Vector Table For Task

The Specify SST Vector Table For Task directive instructs the system to record the address of a table of SST service routine entry points for use by the issuing task.

To deassign the vector table, omit the parameters `adr` and `len` from the macro call.

Whenever an SST service routine entry is specified in both the table used by the task and the table used by a debugging aid, the trap occurs for the debugging aid, not for the task.

#### Fortran Call

Neither the Fortran language nor the ISA standard permits direct linking to system trapping mechanism; therefore, this directive is not available to Fortran tasks.

#### Macro Call

```
SVTK$ [adr][,len]
```

`adr` address of SST vector table

`len` length of (that is, number of entries in) the table in words

The vector table has the following format:

Word 0	odd address of nonexistent memory error
Word 1	memory protect violation
Word 2	T-bit trap or execution of a BPT instruction
Word 3	execution of an IOT instruction
Word 4	execution of a reserved instruction
Word 5	execution of a non-RSX EMT instruction (See Note)
Word 6	execution of a TRAP instruction
Word 7	reserved for future use

A 0 entry in the table indicates that the task does not want to process the corresponding SST.

#### Macro Expansion

```
SVTK$ SSTTBL , 4
.BYTE 105. , 3 ;SVTK$ MACRO DIC, DPB SIZE=3 WORDS
.WORD SSTTBL ;ADDRESS OF SST TABLE
.WORD 4 ;SET TABLE LENGTH=4 WORDS
```

**Local Symbol Definitions**

S.VTTA        table address (2)  
S.VTTL        table length (2)

**DSW Return Codes**

IS.SUC        successful completion  
IE.ADP        part of the DPB or table is out of the issuing task's address  
                 space  
IE.SDP        DIC or DPB size is invalid

**Note**

1. A non-RSX EMT instruction is any EMT instruction not normally used by the system (EMT 1 through 375).

## SWST\$

### 9.1.68 SWST\$—Switch State

The SWST\$ directive makes it possible for a privileged task that is not itself mapped to the Executive to map subroutines that require access to the Executive. The subroutines must be written in position-independent code (PIC). Address references must use absolute mode or PC-relative mode. (See the *PDP-11 MACRO-11 Reference Manual*.)

The SWST\$ directive maps the subroutine through APR5 (that is, it uses virtual addresses 120000 through 137777 octal). Therefore, the subroutine must fall within the limits of 4K words of the base virtual address specified in the directive. The subroutine itself is executed as part of the SWST\$ directive and is, therefore, in system state during its execution. Local data references must also be within the 4K word limit.

#### Fortran Call

There is no Fortran call for the SWST\$ directive.

#### Macro Call

SWST\$\$ base,addr

base	the base virtual address within the task for mapping the subroutine through APR5
addr	virtual address of the subroutine to be executed in system state by the directive

#### Macro Expansion

```

SWST$  BASE,ADDR
.BYTE  175.,3      ;SWST$ MACRO DIC, DPB SIZE = 3 WORDS
.WORD  BASE        ;BASE VIRTUAL ADDRESS FOR MAPPING
                          ;THE SUBROUTINE THROUGH APR5
.WORD  ADDR        ;VIRTUAL ADDRESS OF THE SUBROUTINE
                          ;EXECUTED AT SYSTEM STATE

```

#### Local Symbol Definitions

S.WBAS	base virtual address for mapping the subroutine through APR5
S.WADD	virtual address of the subroutine executed at system state

#### DSW Return Codes

IS.SUC	successful completion
IE.PRI	the issuing task is not privileged



IE.MAP	the specified system state routine is greater than 4K words from the specified base
IE.ADP	part of the DPB is out of the issuing task's address space
IE.SDP	DIC or DPB size is invalid

### Notes

1. User mode register contents are preserved across the execution of the kernel mode subroutine. Contents of the user mode registers are passed into the kernel mode registers. Contents of the kernel mode registers are discarded when the subroutine has completed execution.
2. User mode registers appear at the following octal stack offsets when executing the specified subroutine in kernel mode:

User mode R0 at S.WSR0 Offset on kernal stack  
 User mode R1 at S.WSR1 Offset on kernal stack  
 User mode R2 at S.WSR2 Offset on kernal stack  
 User mode R3 at S.WSR3 Offset on kernal stack  
 User mode R4 at S.WSR4 Offset on kernal stack  
 User mode R5 at S.WSR5 Offset on kernal stack

If you wish to return any register values to the user mode registers, you must store the desired values on the stack using the above offsets.

3. Virtual address values passed to system state in a register must be realigned through kernal APR5. For example, if R5 contains address  $n$ , and the base virtual address in the DPB is 1000(8), the value in R5 must be aligned using the formula:

$$n+120000+\text{base virtual address}$$

Therefore, the resultant value is  $n+121000$ .

4. The system state subroutine should exit by issuing an RTS PC instruction. This causes a successful directive status to be returned as the directive is terminated.

Caution: Keep in mind that the memory management unit rounds the base address to the nearest 32-word boundary.

## UMAP\$

### 9.1.69 UMAP\$—Unmap Address Window

The Unmap Address Window directive unmaps a specified window. After the window has been unmapped, references to the corresponding virtual addresses are invalid and cause a processor trap to occur.

#### Fortran Call

```
CALL UNMAP (iwdb[,ids])
```

**iwdb** an 8-word integer array containing a Window Definition Block (see Section 7.5.2.2)

**ids** directive status

#### Macro Call

```
UMAP$ wdb
```

**wdb** Window Definition Block address

#### Macro Expansion

```
UMAP$ WDBADR
.BYTE 123.,2 ;UMAP$ MACRO DIC, DPB SIZE=2 WORDS
.WORD WDBADR ;WDB ADDRESS
```

Table 9-11  
Window Definition Block Parameters

<i>Input Parameters</i>		
<i>Array Element</i>	<i>Offset</i>	<i>Description</i>
iwdb(1) bits 0-7	W.NID	ID of the window to be unmapped
<i>Output Parameters</i>		
iwdb(7)	W.NSTS	Bit settings <sup>17</sup> in the window status word:
		<i>Bit</i> <i>Definition</i>
	WS.UNM	1 if the window was successfully unmapped

17. If you are a higher-level language programmer, refer to Section 7.5.2 to determine the bit values represented by the symbolic names described.

**Local Symbol Definitions**

U.MABA      Window Definition Block address (2)

**DSW Return Codes**

IS.SUC      successful completion

IE.ITS      the specified address window is not mapped

IE.NVW      invalid address window ID

IE.ADP      DPB or WDB out of range

IE.SDP      DIC or DPB size is invalid

## USTP\$

### 9.1.70 USTP\$—Unstop Task

The Unstop Task directive unstops the specified task that has stopped itself by either the Stop or the Receive Data Or Stop directive. It does not unstop tasks stopped for event flag or tasks stopped for buffered I/O. If the Unstop directive is issued to a task previously stopped by means of the Stop or Receive Or Stop directive while at task state, and the task is presently at AST state, the task only becomes unstopped when it returns to task state.

It is the responsibility of the unstopped task to determine if it has been validly unstopped.

The Unstop directive does not cause a significant event.

#### Fortran Call

```
CALL USTP (rname[,ids])
```

rname            name of task to be unstopped  
ids                integer to receive directive status information

#### Macro Call

```
USTP$ tname
```

tname            name of task to be unstopped

#### Macro Expansion

```
USTP$    ALPHA  
.BYTE    133.,3                ;USTP$ MACRO DIC, DPB SIZE=3 WORDS  
.RAD50   /ALPHA/               ;NAME OF TASK TO BE UNSTOPPED
```

#### Local Symbol Definitions

U.STTN        task name (4)

#### DSW Return Codes

IS.SUC        successful completion  
IE.INS        the specified task is not installed in the system  
IE.ACT        the specified task is not active  
IE.ITS        the specified task is not stopped, or it is stopped for event flag  
              or buffered I/O  
IE.ADP        part of the DPB is out of the issuing task's address space  
IE.SDP        DIC or DPB size is invalid

**VRCD\$****9.1.71 VRCD\$—Variable Receive Data**

The Variable Receive Data directive instructs the system to dequeue a variable-length data block for the issuing task; the data block has been queued (FIFO) for the task by a Variable Send Data directive. When a sender task is specified, only data sent by the specified task is received.

Buffer size can be 256<sub>10</sub> words maximum. If no buffer size is specified, the buffer size is 13<sub>10</sub> words. If a buffer size greater than 256<sub>10</sub> is specified, an IE.IBS error is returned.

A 2-word sender task name (in Radix-50 form) and the data block are returned in the specified buffer, with the task name in the first two words. For this reason, the storage you allocate within the buffer should be two words greater than the size of the data portion of the message specified in the directive.

Variable-length data blocks are transferred from the sending task to the receiving task by means of buffers in the secondary pool.

**Fortran Call**

```
CALL VRCD ([task],bufadr,[buflen],[ids])
```

task	sender task name
buf	address of buffer to receive the sender task name and data
buflen	length of buffer
ids	integer to receive the Directive Status Word.

If the directive was successful, it returns the number of words transferred into the user buffer. If the directive encounters an error during execution, it returns the error code in the ids parameter.

Any error return of the form IE.XXX is a negative word value. If the status is positive, the value of the status word is the number of words transferred including the task name. For example, if you specify a buffer size of 13 in the VRCD\$ call, the value returned in the Directive Status Word is 15 (13 words of data plus the two words needed to return the task name).

**Macro Call**

```
VRCD$ [task],bufadr[,buflen]
```

task	sender task name
bufadr	buffer address
buflen	buffer size in words

### Macro Expansion

```
VRCD$      SNDTSK,DATBUF,BUFSIZ
.BYTE      75.,6          ;VRCD$ MACRO DIC, DPB SIZE=6 WORDS
.RAD50     /SNDTSK/      ;SENDER TASK NAME
.WORD      DATBUF        ;ADDRESS OF DATA BUFFER
```

**VRCS\$****9.1.72 VRCS\$—Variable Receive Data Or Stop**

The Variable Receive Data Or Stop directive instructs the system to dequeue a variable-length data block for the issuing task; the data block has been queued (FIFO) for the task by a Variable Send Data directive. If there is no such packet to be dequeued, the issuing task is stopped. In this case, another task (the sender task) is expected to issue an Unstop directive after sending the data. When stopped in this manner, the directive status returned is IS.SET, indicating that the task was stopped and that no data has been received; however, since the task must be unstopped in order to see this status, the task can now reissue the Variable Receive Data Or Stop directive to actually receive the data packet.

When a sender task is specified, only data sent by the specified task is received.

Buffer size can be  $256_{10}$  words maximum. If no buffer size is specified, the buffer size is  $139_{10}$  words. If a buffer size greater than  $256_{10}$  is specified, an IE.IBS error is returned.

A 2-word sender task name (in Radix-50 form) and the data block are returned in the specified buffer, with the task name in the first 2 words. For this reason, the storage you allocate within the buffer should be two words greater than the size of the data portion of the message specified in the directive.

Variable-length data blocks are transferred from the sending task to the receiving task by means of buffers in the secondary pool.

**Fortran Call**

```
CALL VRCS ([task],bufadr,[buflen][,ids])
```

task	sender task name
buf	address of buffer to receive the sender task name and data
buflen	length of buffer
ids	integer to receive the directive status word

If the directive was successful, it returns the number of words transferred into the user buffer. If the directive execution encountered an error, it returns the error code in the ids parameter.

Any error return of the form IE.XXX is a negative word value. If the status is positive, the value of the status word is the number of words transferred including the taskname. For example, if you specify a buffer size of 13 in the VRCS\$ call, the value returned in the directive status word is 15 (13 words of data plus the two words needed to return the taskname).

**Macro Call**

VRCS\$ [task],bufadr[,buflen]

task            sender task name  
 bufadr         buffer address  
 buflen         buffer size in words

**Macro Expansion**

```

VRCS$         SNDTSK , DATBUF , BUFSIZ
.BYTE         139. , 6                 ;VRCS$ MACRO DIC, DPB SIZE=6 WORDS
.RAD50         /SNDTSK/               ;SENDER TASK NAME
.WORD         DATBUF                 ;ADDRESS OF DATA BUFFER
.WORD         BUFSIZ                 ;BUFFER SIZE
  
```

**Local Symbol Definitions**

R.VSTN         sender task name (4)  
 R.VSBA         buffer address (2)  
 R.VSBL         buffer length (2)  
 R.VSTI         reserved (2)

**DSW Return Codes**

IS.SUC         successful completion  
 IE.INS         specified task not installed  
 IE.RBS         receive buffer is too small  
 IE.IBS         invalid buffer size specified (greater than 255.)  
 IE.ADP         part of the DPB or buffer is out of the issuing task's address  
                  space  
 IE.SDP         DIC or DPB size is invalid



**VRCX\$****9.1.73 VRCX\$—Variable Receive Data Or Exit**

The Variable Receive Data Or Exit directive instructs the system to dequeue a variable-length data block for the issuing task; the data block has been queued (FIFO) for the task by a Variable Send Data directive. When a sender task is specified, only data sent by the specified task is received.

A 2-word sender task name (in Radix-50 form) and the data block are returned in the specified buffer, with the task name in the first two words. For this reason, the storage you allocate within the buffer should be two words greater than the size of the data portion of the message specified in the directive.

If no data has been sent, a task exit occurs. To prevent the possible loss of send data packets, the user should not rely on I/O rundown to take care of any outstanding I/O or open files; the task should assume this responsibility.

Buffer size can be  $256_{10}$  words maximum. If no buffer size is specified, the buffer size is  $13_{10}$  words. If a buffer size greater than  $256_{10}$  is specified, an IE.IBS error is returned.

Variable-length data blocks are transferred from the sending task to the receiving task by means of buffers in the secondary pool.

**Fortran Call**

```
CALL VRCX ([task],bufadr,[buflen][,ids])
```

task	sender task name
buf	address of buffer to receive the sender task name and data
buflen	length of buffer
ids	integer to receive the directive status word

If the directive was successful, it returns the number of words transferred into the user buffer. If the directive execution encountered an error, it returns the error code in the ids parameter.

Any error return of the form IE.XXX is a negative word value. If the status is positive, the value of the status word is the number of words transferred including the taskname. For example, if you specify a buffer size of 13 in the VRCX\$ call, the value returned in the directive status word is 15 (13 words of data plus the two words needed to return the taskname).

**Macro Call**

VRCX\$ [task],bufadr[,buflen]

task            sender task name  
 bufadr        buffer address  
 buflen        buffer size in words

**Macro Expansion**

```
VRCX$        SNDTSK ,DATBUF ,BUFSIZ
.BYTE        77.,6                    ;VRCX$ MACRO DIC, DPB SIZE=6 WORDS
.RAD50       /SNDTSK/                ;SENDER TASK NAME
.WORD        DATBUF                  ;ADDRESS OF DATA BUFFER
.WORD        BUFSIZ                  ;BUFFER SIZE
```

**Local Symbol Definitions**

R.VXTN       sender task name (4)  
 R.VXBA       buffer address (2)  
 R.VXBL       buffer length (2)  
 R.VXTI       reserved (2)

**DSW Return Codes**

IS.SUC       successful completion  
 IE.INS       specified task not installed  
 IE.RBS       receive buffer is too small  
 IE.IBS       invalid buffer size specified (greater than 255.)  
 IE.ADP       part of the DPB or buffer is out of the issuing task's address  
              space  
 IE.SDP       DIC or DPB size is invalid

**VSDA\$****9.1.74 VSDA\$—Variable Send Data**

The Variable Send Data directive instructs the system to queue a variable-length data block for the specified task to receive.

Buffer size can be 256<sub>10</sub> words maximum. If no buffer size is specified, the buffer size is 13<sub>10</sub> words. If a buffer size greater than 256<sub>10</sub> is specified, an IE.IBS error is returned.

When an event flag is specified, a significant event is declared if the directive is successfully executed, and the indicated event flag is set for the sending task.

Variable-length data blocks are transferred from the sending task to the receiving task by buffers in the secondary pool.

**Fortran Call**

```
CALL VSDA (task,bufadr,[buflen],[efn][,ids])
```

task	receiver task name
buf	address of buffer to receive the receiver task name and data
buflen	length of buffer
efn	event flag number
ids	integer to receive the directive status word

**Macro Call**

```
VSDA$ task,bufadr[,buflen][,efn]
```

task	receiver task name
bufadr	buffer address
buflen	buffer size in words

**Macro Expansion**

```
VSDA$      RECTSK ,DATBUF ,BUFSIZ ,4
.BYTE      71. ,8           ;VSDA$ MACRO DIC , DPB SIZE=8 WORDS
.RAD50     /RECTSK/       ;RECEIVER TASK NAME
.WORD      DATBUF         ;ADDRESS OF DATA BUFFER
.WORD      4              ;EVENT FLAG 4
.WORD      BUFSIZ        ;BUFFER SIZE
```

### Local Symbol Definitions

S.DATN	sender task name (4)
S.DABA	buffer address (2)
S.DAEF	event flag number (2)
S.DABL	buffer length (2)
S.DATI	reserved (2)

### DSW Return Codes

IS.SUC	successful completion
IE.UPN	insufficient dynamic storage
IE.INS	specified task not installed
IE.IBS	invalid buffer size specified (greater than 255.)
IE.IEF	invalid event flag number (EFN<0 or EFN>64)
IE.ADP	part of the DPB or buffer is out of the issuing task's address space
IE.SDP	DIC or DPB size is invalid

**VSRC\$****9.1.75 VSRC\$—Variable Send, Request and Connect**

The Variable Send, Request and Connect directive performs a Variable Send Data to the specified task, requests the task if it is not already active, and then connects to the task. The receiver task normally returns status by the Emit Status or the Exit With Status directive.

Buffer size can be  $256_{10}$  words maximum. If no buffer size is specified, the buffer size is  $13_{10}$  words. If a buffer size greater than  $256_{10}$  is specified, an IE.IBS error is returned.

**Fortran Call**

```
CALL VSRC (rname, ibuf,[ibufen],[iefn],[iast],[iesb],
           [iparm],[ids])
```

rname	target task name of the offspring task to be connected
ibuf	name of send buffer
ibufen	length of the buffer
iefn	event flag to be set when the offspring task exits or emits status
iast	name of an AST routine to be called when the offspring task exits or emits status
iesb	name of an 8-word status block to be written when the offspring task exits or emits status
	Word 0          offspring task exit status
	Word 1          system abort code
	Word 2-7        reserved
Note: The exit status block defaults to one word. To use the 8-word exit status block, you must specify the logical OR of the symbol SP.WX8 and the event flag number in the iefn parameter above.	
iparm	name of a word to receive the status block address when an AST occurs
ids	integer to receive the Directive Status Word

**Macro Call**

```
VSRC$ tname,buf[,bufen],[efn],[east],esb]
```

tname	target task name of the offspring task to be connected
buf	address of send buffer
bufen	length of buffer
efn	the event flag to be cleared on issuance and set when the offspring task exits or emits status

east	address of an AST routine to be called when the offspring task exits or emits status
esb	address of an 8-word status block to be written when the offspring task exits or emits status
Word 0	offspring task exit status
Word 1	system abort code
Word 2-7	reserved

Note: The exit status block defaults to 1 one word. To use the 8-word exit status block, you must specify the logical OR of the symbol SP.WX8 and the event flag number in the efn parameter above.

### Macro Expansion

```

VSRC$      ALPHA,BUFFR,BUFSIZE,2,SDRCTR,STBLK
.BYTE      141.,8          ;VSRC$ MACRO DIC, DPB SIZE=8 WORDS
.RAD50     /ALPHA/        ;TARGET TASK NAME
.WORD      BUFR           ;SEND BUFFER ADDRESS
.BYTE      2              ;EVENT FLAG NUMBER = 2
.BYTE      16.           ;EXIT STATUS BLOCK CONSTANT
.WORD      BUFSIZE        ;LENGTH OF BUFFER IN BYTES
.WORD      SDRCTR         ;ADDRESS OF AST ROUTINE
.WORD      STBLK         ;ADDRESS OF STATUS BLOCK

```

### Local Symbol Definitions

S.DRTN	task name (4)
S.DRBF	buffer address (2)
S.DREF	event flag (2)
S.DREA	AST routine address (2)
S.DRES	status block address (2)

### DSW Return Codes

IS.SUC	successful completion
IE.UPN	insufficient dynamic memory to allocate a send packet, Offspring Control Block, Task Control Block, or Partition Control Block
IE.INS	the specified task is an ACP or has the no-send attribute
IE.IEF	an invalid event flag number was specified (EFN<0 or EFN>64)
IE.ADP	part of the DPB or exit status block is not in the issuing task's address space
IE.SDP	DIC or DPB size is invalid

### Notes

1. Changing the virtual mapping of the exit status block while the connection is in effect may result in obscure errors.

**WIMP\$****9.1.76 WIMP\$—What's In My Professional**

The What's In My Professional directive is a general purpose system information retrieval mechanism. The directive allows a nonprivileged task to retrieve specific information stored by the system without requiring the task to be mapped to the Executive. In all forms, the WIMP\$ directive requires a subfunction, a return buffer, and the return buffer size.

A subfunction specifies the type of information to be returned. The return buffer is space allocated within your task and must be large enough to contain the information that is to be returned. Refer to the descriptions of the implemented subfunction for the specific size of the return buffer.

**Fortran Call**

```
CALL WIMP (SFCN,P1,P2,P3,P4,P5,P6,IDS)
```

**Macro Call**

```
WIMP$ SFCN,P1,P2,P3,P4,P5,P6
```

SFCN	subfunction code
P1	parameter 1
P2	parameter 2
P3	parameter 3
P4	parameter 4
p5	parameter 5
P6	parameter 6
IDS	directive status

**Macro Expansion**

```
WIMP$ SFCN,P1,P2,P3,P4,P5,P6
```

```
.BYTE 169.,variable
.WORD SFCN ;SUBFUNCTION CODE
.WORD P1 ;PARAMETER 1
.WORD P2 ;PARAMETER 2
.WORD Pn ;PARAMETER n
```

**Local Symbol Definitions**

G.INSF	SUBFUNCTION CODE (2)
G.IP01	PARAMETER 1 (2)

G.IP02	PARAMETER 2 (2)
G.IP03	PARAMETER 3 (2)
G.IP04	PARAMETER 4 (2)
G.IP05	PARAMETER 5 (2)
G.IP06	PARAMETER 6 (2)

**DSW Return Codes**

IS.SUC	successful completion
IE.IDU	invalid hardware for requested operation
IE.SDP	DIC, DPB size, or subfunction is invalid

**Implemented Subfunctions**

GI.SSN	get system serial number
WIMP\$ GI.SSN,BUF,SIZ	
BUF	return buffer address
SIZ	size in words of return buffer (size = 3) <sub>10</sub>

**Output Buffer Format**

word 0	high word of system serial number
word 1	middle word
word 2	low word
GI.CFG	get configuration table
WIMP\$ GI.CFG,BUF,SIZ	
BUF	return buffer address
SIZ	size in words of return buffer (size = 96) <sub>10</sub>

Table 9-12 lists the offsets in the configuration table as displayed in the user's return buffer. The information contained in the return buffer reflects the current system configuration including hardware and hardware status. Any changes made to the information in the return buffer are not reflected in the system configuration table.



Table 9-12  
The Configuration Table Output Buffer Format

<i>Description</i>	<i>Offset</i>
Table length in bytes	0
Serial number ROM ID	2
High word of serial number	4
Middle word of serial number	6
Low word of serial number	8.
Number of option slots	10.
Data length of table	12.
Slot 0 ID	14.
Status/error of slot 0	16.
Slot 1 ID	18.
Status/error of slot 1	20.
Slot 2 ID	22.
Status/error of slot 2	24.
Slot 3 ID	26.
Status/error of slot 3	28.
Slot 4 ID	30.
Status/error of slot 4	32.
Slot 5 ID	34.
Status/error of slot 5	36.
Slot 6 ID (not used)	38.
Status/error of slot 6 (not used)	40.
Slot 7 ID (not used)	42.
Status/error of slot 7 (not used)	44.
Keyboard ID (supplied by the keyboard, this could be some other input device)	46.
Keyboard status/error	48.
Base processor type	50.
Base processor status	52.
Primary memory ID	54.
Total system memory size	56.
Diagnostic ROM version number	58.
Diagnostic ROM error status	60.
Video monitor present	62.
Video monitor status	64.
Audio device ID (not on PRO 325/350)	66.
Audio device status	68.
Keyboard interface ID (2661)	70.
Keyboard interface status/error	72.
Printer port interface ID (2661)	74.

Table 9-2 (Cont.)

<i>Description</i>	<i>Offset</i>
Printer port interface status/error	76.
Maintenance port ID	78.
Maintenance port status	80.
Serial comm interface ID	82.
Serial comm interface status/error	84.
Time of day device ID	86.
Time of day status/error	88.
NVR RAM ID	90.
NVR RAM status/error	92.
Floating point ID	94.
Floating point status/error	96.
Interrupt controller ID	98.
Interrupt controller status/error	100.
Reserved locations	102.
Reserved locations	104.
Reserved locations	106.
Reserved locations	108.
Reserved locations	110.
Reserved locations	112.
Reserved locations	114.
Reserved locations	116.
Reserved locations	118.
Reserved locations	120.
Reserved locations	122.
Reserved locations	124.
Reserved locations	126.
Reserved locations	128.
Reserved locations	130.
Reserved locations	132.
Soft restart address	134.
Offset value into boot code	136.
Booted device ID number	138.
Unit number of booted device	140.
Current boot sequence return address	142.
Error flag for ROM diagnostics	144.
Additional information length	146.

**WSIG\$****9.1.77 WSIG\$—Wait For Significant Event (\$S Form Recommended)**

The Wait For Significant Event directive is used to suspend the execution of the issuing task until the next significant event occurs. It is an especially effective way to block a task that cannot continue because of a lack of dynamic memory, since significant events occurring throughout the system often result in the release of dynamic memory. The execution of a Wait For Significant Event directive does not itself constitute a significant event.

**Fortran Call**

```
CALL WFSNE
```

**Macro Call**

```
WSIG$S [err]
```

err     error routine address

**Macro Expansion**

```
WSIG$S  ERR
MOV      (PC)+, -(SP)      ;PUSH DPB ONTO THE STACK
.BYTE    49., 1            ;WSIG$S MACRO DIC, DPB SIZE=1 WORD
EMT      377                ;TRAP TO THE EXECUTIVE
BCC      .+6                ;BRANCH IF DIRECTIVE SUCCESSFUL
JSR      PC,ERR             ;OTHERWISE, CALL ROUTINE ''ERR''
```

**Local Symbol Definitions**

None

**DSW Return Codes**

IS.SUC	successful completion
IE.ADP	part of the DPB is out of the issuing task's address space
IE.SDP	DIC or DPB size is invalid

**Notes**

1. If a directive is rejected for lack of dynamic memory, this directive is the only technique available for blocking task execution until dynamic memory may again be available.
2. The wait state induced by this directive is satisfied by the first significant event to occur after the directive has been issued. The significant event that occurs may or may not be related to the issuing task.

3. Because this directive requires only a 1-word DPB, the \$\$ form of the macro is recommended. It requires less space and executes with the same speed as that of the DIR\$ macro.
4. Significant events include the following:
  - I/O completion
  - Task exit
  - Execution of a Send Data directive
  - Execution of a Send Data, Request and Pass OCB directive
  - Execution of a Send, Request and Connect directive
  - Execution of a Send By Reference directive or a Receive by Reference directive
  - Execution of an Alter Priority directive
  - Removal of an entry from the clock queue (for instance, resulting from the execution of a Mark Time directive or the issuance of a rescheduling request)
  - Execution of a Declare Significant Event directive
  - Execution of the round-robin scheduling algorithm at the end of a round-robin scheduling interval
  - Execution of an Exit, an Exit with Status, or Emit Status directive

**WTLO\$****9.1.78 WTLO\$—Wait For Logical OR Of Event Flags**

The Wait For Logical OR Of Event Flags directive instructs the system to block the execution of the issuing task until the Executive sets the indicated event flags from one of the following groups:

- GR 0 flags 1-16
- GR 1 flags 17-32
- GR 2 flags 33-48
- GR 3 flags 49-64

The task does not block itself if any of the indicated flags are already set when the task issues the directive. See Notes below.

**Fortran Call**

```
CALL WFLOR (efn1,efn2,...efnn)
```

efn list of event flag numbers taken as the set of flags to be specified in the directive

**Macro Call**

```
WTLO$ grp,msk
```

grp desired group of event flags

msk a 16-bit flag mask word

**Macro Expansion**

```
WTLO$ 2,160003
.BYTE 43.,3 ;WTLO$ MACRO DIC, DPB SIZE=3 WORDS
.WORD 2 ;FLAGS SET NUMBER 2 (FLAGS 33:48.)
.WORD 160003 ;EVENT FLAGS 33,34,46,47 AND 48.
```

**Local Symbol Definitions**

None

**DSW Return Codes**

- IS.SUC successful completion
- IE.IEF no event flag specified in the mask word or flag group indicator other than 0, 1, 2, 3, 4, or 5
- IE.ADP part of the DPB is out of the issuing task's address space
- IE.SDP DIC or DPB size is invalid

**Notes**

1. There is a one-to-one correspondence between bits in the mask word and the event flags in the specified group. That is, if group 1 were specified, then bit 0 in the mask word would correspond to event flag 17, bit 1 to event flag 18, and so forth.
2. The Executive does not arbitrarily clear event flags when Wait For conditions are met. Some directives (Queue I/O Request, for example) implicitly clear a flag; otherwise, they must be explicitly cleared by a Clear Event Flag directive.
3. The grp operand must always be of the form n regardless of the macro form used. In all other macro calls, numeric or address values for \$\$ form macros have the form:  
#n  
For WTLO\$\$ this form of the grp argument would be:  
n
4. The argument list specified in the FORTRAN call must contain only event flag numbers that lie within one event flag group. If event flag numbers are specified that are in more than one group, or if an invalid event flag number is specified, a fatal FORTRAN error is generated.
5. If the issuing task has outstanding buffered I/O when it enters the Wait For state, it will be stopped. When the task is in a stopped state, it can be checkpointed by any other task regardless of priority. The task is unstopped when:
  - The outstanding buffered I/O completes.
  - The Wait For condition is satisfied.
  - The issuing task exits before the Wait For condition is satisfied.

**WTSE\$****9.1.79 WTSE\$—Wait For Single Event Flag**

The Wait For Single Event Flag directive instructs the system to block the execution of the issuing task until the indicated event flag is set. If the flag is set at issuance, task execution is not blocked.

**Fortran Call**

```
CALL WAITFR (efn[,ids])
```

efn event flag number

ids directive status

**Macro Call**

```
WTSE$efn
```

efn event flag number

**Macro Expansion**

```
WTSE$ 52.
.BYTE 41.,2 ;WTSE$ MACRO DIC, DPB SIZE=2 WORDS
.WORD 52. ;EVENT FLAG NUMBER 52.
```

**Local Symbol Definitions**

W.TSEF event flag number (2)

**DSW Return Codes**

IS.SUC successful completion

IE.IEF invalid event flag number (EFN<1, or EFN>64)

IE.ADP part of the DPB is out of the issuing task's address space

IE.SDP DIC or DPB size is invalid

**Notes**

1. If the issuing task has outstanding buffered I/O when it enters the Wait For state, it will be stopped. When the task is in a stopped state, it can be checkpointed by any other task regardless of priority. The task is unstopped when:
  - The outstanding buffered I/O completes.
  - The Wait For condition is satisfied.
2. The issuing task exits before the Wait For condition is satisfied.





## CHAPTER 10

# SYSTEM INPUT/OUTPUT CONVENTIONS

---

This chapter describes the device drivers supported by the system and the characteristics, functions, error conditions, and programming hints associated with each one. Devices not described in this chapter must be developed and maintained by the user.

Input/output (I/O) operations on the system are extremely flexible and are as device- and function-independent as possible. Programs issue I/O requests to logical units that have been previously associated with particular physical device units. Each program or task is able to establish its own correspondence between physical device units and logical unit numbers (LUNs). I/O requests are queued as issued; they are subsequently processed according to the relative priority of the tasks that issued them. I/O requests (for appropriate devices) can be issued from tasks by means of either the Record Management Services (RMS) or can be interfaced directly to an I/O driver by means of the Queue I/O (QIO) system directive.

All of the I/O services described in this chapter are requested by the user in the form of QIO system directives. A function code included in the QIO directive indicates the particular input or output operation to be performed. I/O functions can be used to request such operations as:

- Attaching or detaching a physical device unit for a task's exclusive use
- Reading or writing a logical or virtual block of data
- Cancelling a task's I/O requests

A wide variety of device-specific I/O operations (for example, reading from a terminal without echoing characters) can also be specified with QIO directives.

## 10.1 PHYSICAL, LOGICAL, AND VIRTUAL I/O

There are three possible modes in which an I/O transfer can take place: physical, logical, and virtual.

Physical I/O involves reading and writing data in the actual physical units used by the hardware (sectors or blocks).

Logical I/O involves reading and writing data in units (blocks) used by software. When you issue a QIO to a device driver, the driver translates the logical block numbers to physical block numbers. Logical blocks are numbered beginning at 0, and are always 512<sub>10</sub> bytes in length.

Virtual I/O also involves reading and writing data in units (blocks) used by software. However, virtual I/O pertains to reading and writing data in open files. When reading and writing data in file-structured devices such as disks, virtual blocks are the same size as logical blocks, but are numbered starting at 1 instead of 0. When you issue a QIO to read or write a virtual block in an open file, the system translates virtual blocks into logical blocks. When you issue a QIO to read or write a virtual block to a non-file-structured device such as a terminal, the Executive changes the QIO from a read/write virtual block to a read/write logical block.

## 10.2 SUPPORTED DEVICES

The system supports the devices listed below when they are connected to the printer port. Drivers are supplied for each of these devices; the appropriate I/O operations are described in detail in subsequent chapters of this manual.

### 1. Terminals

- LA12 DECwriter
- LA34/LA38 DECwriter IV
- LA100 DECwriter
- LA120
- VT100 Alphanumeric Display Terminal
- VT101 Alphanumeric Display Terminal
- VT102 Alphanumeric Display Terminal
- VT105 Alphanumeric Display Terminal
- VT125 Alphanumeric Display Terminal
- VT131 Alphanumeric Display Terminal
- VT132 Alphanumeric Display Terminal

### 2. Disks

- RD50 Fixed 5-Megabyte Hard Disk
- RX50 5¼-inch Diskette

### 10.3 LOGICAL UNITS

This section describes the construction of the logical unit table and the use of logical unit numbers.

#### 10.3.1 Logical Unit Number

A logical unit number, or LUN, is a number associated with a physical device unit during system I/O operations. More simply, a LUN represents an association between a logical unit and a physical device unit. For example, LUN 1 might be associated with the terminal, LUN 2 with the printer port, LUNs 3 and 4 with the RX50s, and LUN5 with the RD50. Once the association has been made, the LUN provides a direct and efficient mapping to the physical device unit, and eliminates the necessity to search the device tables whenever the system encounters a reference to a physical device unit.

The association is a dynamic one; each task running in the system can establish its own correspondence between LUNs and physical device units, and can change any LUN/physical-device-unit association at almost any time. The flexibility of this association contributes heavily to system device independence.

Keep in mind that, although this association can be changed at any time, reassignment of a LUN at run time causes pending I/O requests for the previous LUN assignment to be cancelled. It is the user's responsibility to verify that all outstanding I/O requests for a LUN have been serviced before that LUN is associated with another physical device unit.

#### 10.3.2 Logical Unit Table

There is one Logical Unit Table (LUT) for each task running in a system. This table is a variable-length block contained in the task header. Each LUT contains sufficient 2-word entries for the number of logical units specified by the user at task-build time by the "UNITS=" option.

Each entry or slot contains a pointer to the physical device unit currently associated with that LUN. Whenever a user issues an I/O request, the system matches the appropriate physical device unit to the LUN specified in the call by indexing into the LUT by the number supplied as the LUN. Thus, if the call specifies 6 as the LUN, the system accesses the sixth 2-word entry in the LUT and associates the I/O request with the physical device unit to which the entry points. The number of LUN assignments valid for a task ranges from 0 to 255, but cannot be greater than the number of LUNs specified at task-build time.

### 10.3.3 Changing LUN Assignments

Logical unit numbers have no significance until they are associated with a physical device unit by means of one of the methods described below:

1. At task-build time, the user can specify an ASG= keyword option, which associates a physical device unit with a logical unit number referenced in the task being built.
2. At run time, a task can dynamically change a LUN assignment by issuing the Assign LUN system directive, which changes the association of a LUN with a physical device unit during task execution.

## 10.4 ISSUING AN I/O REQUEST

User tasks perform I/O in the system by submitting requests for I/O service in the form of QIO or QIO And Wait system directives.

In this system, and in most multiprogramming systems, tasks normally do not directly access physical device units. Instead, they utilize input/output services provided by the Executive, since the Executive can effectively multiplex the use of physical device units over many users. The Executive routes I/O requests to the appropriate device driver and queues them according to the priority of the requesting task. I/O operations proceed concurrently with other activities in the system.

Before a request is queued, it must pass a battery of acceptance tests administered by the Executive. If the request fails, it is rejected; this rejection is signaled by the setting of the C-bit when the statement following the QIO is executed. It is good programming practice to check for directive rejection by following the QIO directive with a BCS instruction.

After an I/O request has been queued, the system does not wait for the operation to complete. If at any time the user task that issued the QIO request cannot proceed until the I/O operation has completed, it should specify an event flag (see Section 5.2) in the QIO request and should issue a Wait For system directive specifying the same event flag at the point where synchronization must occur. The task then waits for completion of I/O by waiting for the specified event flag to be set.

The QIOW directive, QIO And Wait, is a more economical way to achieve this synchronization. QIOW automatically waits until I/O has completed before returning control to the task. Thus, the additional Wait For directive is not necessary.

Each QIO or QIOW directive must supply sufficient information to identify and queue the I/O request. The user may also want to include locations to receive error or status codes and to specify the address of an asynchronous system

trap service routine. Certain types of I/O operations require the specification of device-dependent information as well. Typical QIO parameters are the following:

- I/O function to be performed
- Logical unit number associated with the physical device unit to be accessed
- Optional event flag number for synchronizing I/O completion processing (required for QIOW)
- Optional address of the I/O status block to which information indicating successful or unsuccessful completion is returned
- Optional address of an asynchronous system trap service routine to be entered on completion of the I/O request
- Optional device- and function-dependent parameters specifying such items as the starting address of a data buffer, the size of the buffer, and a block number

A set of system macros that facilitate the issuing of QIO directives is supplied with the system. These macros, which reside in the System Library Account in (RSXMAC.SML), must be made available to the source program by means of the MACRO-11 Assembler directive `.MCALL`. The function of `.MCALL` is described in Section 10.6.4. Several of the first six parameters in the QIO directive are optional, but space for these parameters must be reserved.

During expansion of a QIO macro, a value of 0 is defaulted for all null (omitted) parameters. Inclusion of the device- and function-dependent parameters depends on the physical device unit and function specified. If the user wanted to specify only an I/O function code, a LUN, and an address for an asynchronous system trap service routine, the following might be issued:

```
QIO$C IO.ATT,6,,ASTOX
```

IO.ATT	The I/O function code for attach
6	The LUN
ASTOX	The AST address
,,,	Null arguments for the event flag number, the request priority, and the address of the I/O status block

No additional device- or function-dependent parameters are required for an attach function. The C form of the QIO\$ macro is used here.

For convenience, any comma may be omitted if no parameters appear to the right of it. Therefore, the command above could be issued as follows, if the asynchronous system trap was not desired:

```
QIO$C IO.ATT,6
```

All extra commas have been dropped. If, however, a parameter appears to the right of any place-holding comma, that comma must be retained.

#### 10.4.1 QIO Macro Format

The arguments for a specific QIO macro call may be different for each I/O device accessed and for each I/O function requested. The general format of the call is, however, common to all devices and is as follows:

```
QIO$C fnc,lun,[efn],[pri],[isb],[ast][,<p1,p2,...,p6>]
```

where brackets ([ ]) enclose optional or function-dependent parameters. If function-dependent parameters <p1,...,p6> are required, these parameters must be enclosed within angle brackets (<>). The following paragraphs summarize the use of each QIO parameter.

The fnc parameter is a symbolic name representing the I/O function to be performed. This name is of the form

```
IO.xxx
```

xxx Identifies the particular I/O operation

For example, a QIO request to attach the physical device unit associated with a LUN specifies the function code

```
IO.ATT
```

A QIO request to cancel (or kill) all I/O requests for a specified LUN begins in the following way:

```
QIO$C IO.KIL,...
```

The fnc parameter specified in the QIO request is stored internally as a function code in the high-order byte and modifier bits in the low-order byte of a single word. The function code is in the range 0 through 31 and is a binary value supplied by the system to match the symbolic name specified in the QIO request. The correspondence between global symbolic names and function codes is defined in the system object module library, which is automatically searched by the Task Builder. Local symbolic definitions may also be obtained by the FILIO\$ and SPCIO\$ macros, which reside in the System Macro Library and are summarized in Appendix C. Several similar functions may have identical function codes, and may be distinguished only by their modifier bits. Only the modifier bits for these two operations are stored differently.

The lun parameter represents the logical unit number (LUN) of the associated physical device unit to be accessed by the I/O request. The association between the physical device unit and the LUN is specific to the task that issues the I/O request, and the LUN reference is usually device independent. An attach request to the physical device unit associated with LUN 14 begins in the following way:

```
QIO$C IO.ATT,14,....
```

Because each task has its own LUT in which the physical device unit-LUN correspondences are established, the legality of a LUN parameter is specific to the task that includes this parameter in a QIO request. In general, the LUN must be in the following range:

```
0 <LUN <length of task's LUT (if nonzero)
```

The number of LUNs specified in the LUT of a particular task cannot exceed 255.

The efn parameter is a number representing the event flag to be associated with the I/O operation. It may optionally be included in a QIO or QIO And Wait request. The specified event flag is cleared when the I/O request is queued and is set when the I/O operation has completed. If the task has issued the QIO And Wait directive, execution is automatically suspended until the I/O completes. If a QIO directive has been issued (with no Wait For directive), then task execution proceeds in parallel with the I/O. When the task continues to execute, it may test the event flag whenever it chooses by using the Read All Event Flags system directive or the Read Extended Flags system directive (for all event flags) If the user specifies an event flag number, this number must be in the range 1 through 64. If an event flag specification is not desired, efn can be omitted or can be supplied with a value of 0. Event flags 1 through 32 are local (specific to the issuing task); event flags 33 through 64 are global (shared by all tasks in the system). Flags 25 through 32 and 57 through 64 are reserved for use by system software. Within these bounds, the user can specify event flags as desired to synchronize I/O completion and task execution. Chapter 4 provides a more detailed explanation of event flags and significant events.

Note: If an event flag is not specified, the Executive treats the directive as if it were a simple QIO request.

A I/O request automatically assumes the priority of the requesting task. Thus, it is recommended that a value of 0 (or a null) be used for this parameter.

The optional isb parameter identifies the address of the I/O status block (I/O status double-word) associated with the I/O request. This block is a 2-word array in which a code representing the final status of the I/O request is returned on completion of the operation. This code is a binary value that corresponds to a symbolic name of the form IS.xxx (for successful returns) or IE.xxx (for error returns). The binary error code is returned to the low-order byte of the first word

of the status block. It can be tested symbolically, by name. For example, the symbolic status IE.BAD is returned if a bad parameter is encountered. The following illustrates the examination of the I/O status block, IOST, to determine if a bad parameter has been detected:

```

QIO$C   IO.ATT,14.,2.,,IOST
BCS     DIRERR
WTSE$C  2
        .
        .
        .
CMPB    #IS.SUC,IOST
BNE     ERROR

```

The correspondence between global symbolic names and I/O completion codes is defined in the system object module library, which is automatically searched by TKB.

Certain device-dependent information is returned to the high-order byte of the first word of isb on completion of the I/O operation. If a read or write operation is successful, the second word is also significant. For example, in the case of a read function on a terminal, the number of bytes typed before a carriage return is returned in the second word of isb. If a magtape unit is the device and a write function is specified, this number represents the number of bytes actually transferred. The status block can be omitted from a QIO request if the user does not intend to test for successful completion of the request.

The optional ast parameter specifies the address of a service routine to be entered when an asynchronous system trap occurs. Section 10.4.3 discusses the use of asynchronous system traps, and Chapter 5 describes traps in detail. If you want to interrupt a task to execute special code on completion of an I/O request, an asynchronous system trap routine can be specified in the QIO request. When the specified I/O operation completes, control branches to this routine at the software priority of the requesting task. The asynchronous code beginning at address ast is then executed, much as an interrupt service routine would be. If the user does not want to perform asynchronous processing, the ast parameter can be omitted or a value of 0 specified in the QIO macro call.

The additional QIO parameters, <p1,p2,...,p6>, are dependent on the particular function and device specified in the I/O request. Typical parameters may include I/O buffer address, I/O buffer length, and so forth. Between zero and six parameters can be included, depending on the particular I/O function.

## 10.4.2 Significant Events

“Significant event” is a term used in real-time systems to indicate a change in system status. The system declares a significant event when an I/O operation completes. This signals the system that a change in status has occurred and indicates that the Executive should review the eligibility of all tasks in the system to determine which task should run next. The use of significant events helps cooperating tasks in a real-time system to communicate with each other, and thus allows these tasks to control their own sequence of execution dynamically.



Significant events are normally set by system directives, either directly or indirectly, by completion of a specified function. Event flags associated with tasks may be used to indicate which significant event has occurred. Of the 64 event flags available, the flags numbered 1 through 32 are local to an individual task and are set or reset only as a result of that task's operation. The event flags numbered 33 through 64 are common to all tasks. Flags 25 through 32 and 57 through 64 are reserved for system software use.

An example of the use of significant events follows. A task issues a QIO directive with an efn parameter specified. A Wait For directive follows the QIO and specifies as an argument the same event flag number. The event flag is cleared when the I/O request is queued by the Executive, and the task is blocked when it executes the Wait For directive until the event flag is set and a significant event is declared at the completion of the I/O request. The task resumes when the appropriate event flag is set, and execution resumes at the instruction following the Wait For directive. During the time that the task is blocked, other tasks have a chance to run, thus increasing throughput in the system.

### 10.4.3 System Traps

System traps are used to interrupt task execution and to cause a transfer of control to another memory location for special processing. Traps are handled by the Executive and are relevant only to the task in which they occur. To use a system trap, a task must contain a trap service routine, which is automatically entered when the trap occurs.

There are two types of system traps: synchronous and asynchronous. Both are used to handle error or event conditions, but the two traps differ in their relation to the task that is running when they are detected. Synchronous traps signal error conditions within the executing task. If the same instruction sequence were repeated, the same synchronous trap would occur at the same place in the task. Asynchronous traps signal the completion of an external event such as an I/O operation. An asynchronous system trap (AST) usually occurs as the result of initiating or completing an external event rather than a program condition.

The Executive queues ASTs in a FIFO queue for each task and monitors all asynchronous service routine operations. Because asynchronous traps may be the end result of I/O-related activity, they cannot be controlled directly by the task that receives them. However, the task may, under certain circumstances, block recognition of ASTs to prevent simultaneous access to a critical data region. When access to the critical data region has been completed, the queued ASTs may again be honored. The DSAR\$\$ (Disable AST Recognition) and ENAR\$\$ (Enable AST Recognition) system directives provide the mechanism for accomplishing this. An example of an asynchronous trap condition is the completion of an I/O request. The timing of such an operation clearly cannot be predicted by the requesting task. If an AST service routine is not specified in an I/O request, a trap does not occur and normal task execution continues.

Asynchronous system traps associated with I/O requests enable the requesting task to be truly event driven. The AST service routine contained in the initiating task is executed as soon as possible, consistent with the task's priority. Using the AST routine to service I/O-related events provides a response time that is considerably better than a polling mechanism, and provides for better overlap processing than the simple QIO and Waitfor sequence. Asynchronous system traps also provide an ideal mechanism for use in multiple buffering of I/O operations.

All ASTs are inserted in a FIFO queue on a per-task basis as they occur that is, the event that they are to signal has expired; they are effected one at a time whenever the task does not have ASTs disabled and is not already in the process of executing an AST service routine. The process of effecting an AST involves storing certain information on the task's stack, including the task's Wait For mask word and address, the Directive Status Word (DSW), the PS, the PC and any trap dependent parameters. The task's general-purpose registers R0-R5 are not saved, and thus it is the responsibility of the AST service routine to save and restore the registers it uses. After an AST is processed, the trap-dependent parameters (if any) must be removed from the task's stack and an AST Service Exit directive executed. The ASTX\$\$ macro described in Section 10.6.7 is used to issue the AST Service Exit directive. On AST service exit, control is returned to another queued AST, to the executing task, or to another task that has been waiting to run. Chapter 5 describes in detail the purpose of AST service routines and all system directives used to handle them.

## 10.5 DIRECTIVE PARAMETER BLOCKS

A Directive Parameter Block (DPB) is a fixed-length area of contiguous memory that contains the arguments specified in a system directive macro call. The DPB for a QIO directive has a length of 12 words. It is generated as the result of expanding a QIO macro call. The first byte of the DPB contains the directive identification code (DIC)—always 1 for QIO. The second byte contains the size of the DPB in words—always 12. During assembly of a user task containing QIO requests, the MACRO-11 Assembler generates a DPB for each I/O request specified in a QIO macro call. At run time, the Executive uses the arguments stored in each DPB to create, for each request, an I/O packet in system dynamic storage. The packet is entered by priority into a queue of I/O requests for the specified physical device unit. This queue is created and maintained by the Executive and is ordered by the priority of the tasks that issued the requests. The I/O drivers examine their respective queues for the I/O request with the highest priority capable of being executed. This request is dequeued (removed from the queue) and the I/O operation is performed. The process is then repeated until the queue is emptied of all requests.

After the I/O request has been completed, the Executive declares a significant event and may set an event flag, cause a branch to an asynchronous system trap service routine, and/or return the I/O status, depending on the arguments specified in the original QIO macro call. Figure 10-1 illustrates the layout of a sample DPB.

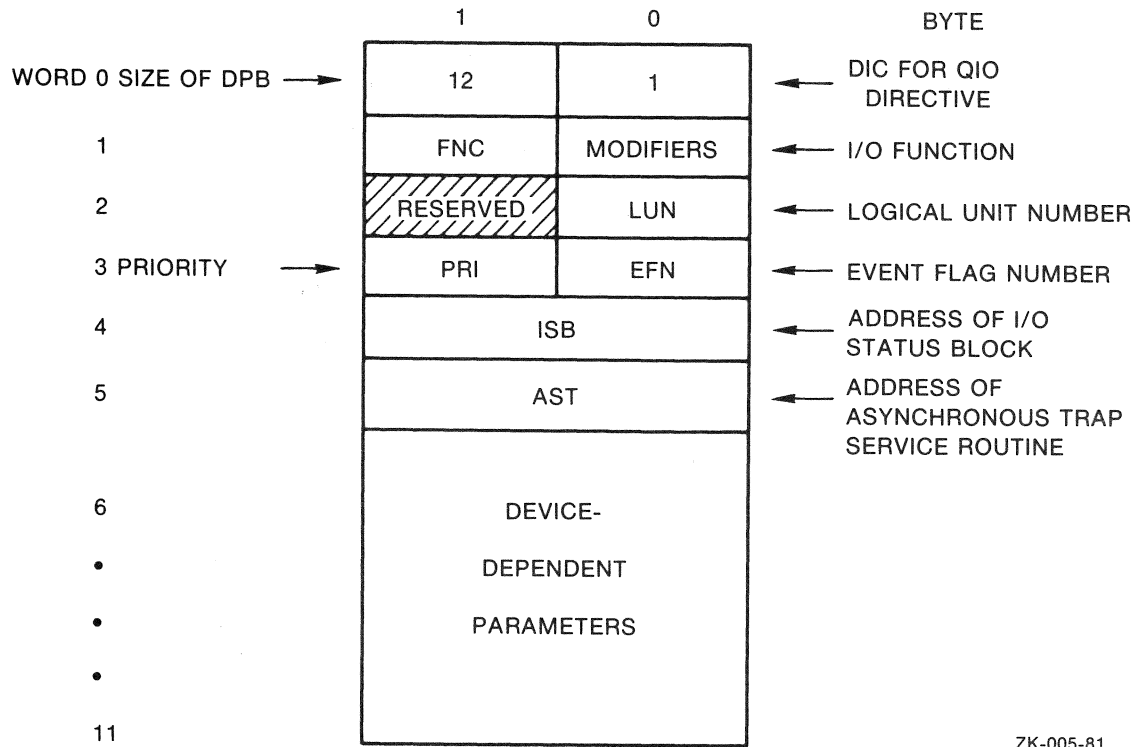


Figure 10-1 QIO Directive Parameter Block

ZK-005-81

## 10.6 I/O-RELATED MACROS

Several system macros are supplied with the system to issue and return information about I/O requests. These macros reside in the System Macro Library and must be made available during assembly by the MACRO-11 assembler directive `.MCALL`.

Also supplied are Fortran-callable subroutines that perform the same functions as the system macros. See Chapter 3 for details.

There are three distinct forms of most of the system directive macros discussed in this section. The following list summarizes the forms of `QIO$`, but the characteristics of each form also apply to `QIOW$`, `ALUN$`, `GLUN$`, and other system directive macros described below.

1. `QIO$` generates a directive parameter block for the I/O request at assembly time, but does not provide the instructions necessary to execute the request. This form of the request is actually executed using the `DIR$` macro. It is useful if the DPB is to be used in several different places in the task and/or modified or referenced by the task at run time.
2. `QIO$$` generates a directive parameter block for the I/O request on the stack, and also generates code to execute the request. This is a useful form for reentrant, shareable code since the DPB is generated dynamically at execution time.

3. QIO\$C generates a directive parameter block for the I/O request at assembly time, and also generates code to execute the request. The DPB is generated in a separate program section called \$DPB\$. This approach incurs little system overhead and is useful when an I/O request is executed from only one place in the program.

Parameters for both the QIO\$ and QIO\$C forms of the macro must be valid expressions to be used in assembler data-generating directives such as .WORD and .BYTE. Parameters for the QIO\$\$ form must be valid source operand address expressions to be used in assembler instructions such as MOV and MOV.B. The following example references the same parameters in the three distinct forms of the macro call.

```

QIO$      ID.RLB,6,2,,,AST01,<RDBUF,80.>
QIO$C     ID.RLB,6,2,,,AST01,<RDBUF,80.>
QIO$$     #ID.RLB,#6,#2,,,#AST01,<#RDBUF,#80.>

```

Only the QIO\$\$ form of the macro produces the DPB dynamically. The other two forms generate the DPB at assembly time.

The following Executive directives and assembler macros are described in this section:

1. QIO\$, which is used to request an I/O operation and supply parameters for that request
2. QIOW\$, which is equivalent to QIO\$ followed by WTSE\$
3. DIR\$, which specifies the address of a directive parameter block as its argument, and generates code to execute the directive
4. .MCALL, which is used to make available from the System Macro Library all macros referenced during task assembly
5. ALUN\$, which is used to associate a logical unit number with a physical device unit at run time
6. GLUN\$, which requests that the information about a physical device unit associated with a specified LUN be returned to a user-specified buffer
7. ASTX\$\$, which is used to terminate execution of an asynchronous system trap (AST) service routine
8. WTSE\$, which instructs the system to block execution of the issuing task until a specified event flag is set

### 10.6.1 The QIO\$ Macro: Issuing an I/O Request

As described in Chapter 3, there are three distinct forms of the QIO\$ macro. QIO\$\$ generates a DPB for the I/O request on the stack, and also generates code to execute the request. QIO\$C generates a DPB and code, but the DPB is generated in a separate program section. QIO\$ generates only the DPB for the I/O request. This form of the macro call is used in conjunction with DIR\$ to ex-

ecute an I/O request. In the following example, the DIR\$ macro actually generates the code to execute the QIO\$ directive. It provides no QIO parameters of its own, but references the QIO directive parameter block at address QIOREF by supplying this label as an argument.

```

QIOREF:  QIO$      IO.RLB,6,2,,,AST01,<BUFFER,80.>
        .
        .
        .
        .
        .
        .
READ1:   DIR$      #QIOREF                               ;ISSUE I/O REQUEST
        .
        .
        .
        .
        .
        .
READ2:   DIR$      #QIOREF                               ;ISSUE I/O REQUEST

```

### 10.6.2 The QIOW\$ Macro: Issuing an I/O Request and Waiting for an Event Flag

The QIOW\$ macro is equivalent to a QIO\$ followed by a WTSE\$. It is more economical to issue a QIO And Wait request than to use the two separate macros. An event flag (efn parameter) must be specified with QIOW\$ if you actually want to wait.

### 10.6.3 The DIR\$ Macro: Executing a Directive

The DIR\$ (execute directive) macro has been implemented to allow a task to reference a previously defined DPB. It is issued in the form:

```
DIR$ [addr][,err]
```

- addr The address of a directive parameter block to be used in the directive. If addr is not included, the DPB itself or the address of the DPB is assumed to be on the stack already. This parameter must be a valid source operand for a MOV instruction generated by the DIR\$ macro.
- err An optional argument which specifies the address of an error routine to which control branches if the directive is rejected. The branch occurs by means of a JSR PC, err if the C-bit is set, indicating rejection of the QIO directive.

### 10.6.4 The .MCALL Directive: Retrieving System Macros

.MCALL is a MACRO-11 Assembler directive that retrieves macros from the System Library Account (in RSXMAC.SML) for use during assembly. It must be included in every user task invoking system macros. .MCALL is usually placed at the beginning of a user-task source module and specifies, as arguments in the call, all system macros that must be made available from the library.

The following example illustrates the use of this directive:

```

.MCALL QIO$,QIO$$S,DIR$,WTSE$$S ;MAKE MACROS AVAILABLE
.
.
.
ATTACH: QIO$$S #IO.ATT,#6,,,IOSB,#AST02 ;ATTACH DEVICE
.
.
.
QIOREF: QIO$$ IO.RLB,6,,,IOSB,AST01,... ;CREATE ONLY QIO DPB
.
.
.
READ1: DIR$ #QIOREF,DIRERR ;ISSUE I/O REQUEST
.
.
.

```

As many macro references as can fit on a line can be included in a single .MCALL directive. There is no limit to the number of .MCALL directives that can be specified.

### 10.6.5 The ALUN\$ Macro: Assigning a LUN

The Assign LUN macro is used to associate a logical unit number with a physical device unit at run time. All three forms of the macro call may be used. Assign LUN does not request I/O for the physical device unit, nor does it attach the unit for exclusive use by the issuing task. It simply establishes a LUN/physical device unit relationship, so that when the task requests I/O for that particular LUN, the associated physical device unit is referenced. The macro is issued from a MACRO-11 program in the following way:

```
ALUN$ lun,dev,unt
```

- lun    The logical unit number to be associated with the specified physical device unit.
- dev    The name of the physical device or a logical device name assigned to a physical device.
- unt    The unit number of the device specified above.

For example, to associate LUN 10 with terminal unit 1, the following macro call could be issued by the task:

```
ALUN$C 10.,TT,1
```

The example included below illustrates the use of the three forms of the ALUN\$ macro.

```

;
; DATA DEFINITIONS
;
ASSIGN: ALUN$ 10.,TT,2 ; GENERATE DPB
.
.
.
;
; EXECUTABLE SECTION
;
DIR$ #ASSIGN ; EXECUTE DIRECTIVE
.
.
.
ALUN$C 10.,TT,2 ; GENERATE DPB IN SEPARATE PROGRAM
. ; SECTION, THEN GENERATE CODE TO
. ; EXECUTE THE DIRECTIVE
.
ALUN$S #10.,#''TI,#0 ; GENERATE DPB ON STACK, THEN
; EXECUTE DIRECTIVE

```

**10.6.5.1 Physical Device Names** — Table 10-1 contains physical device names, listed alphabetically, that may be included as dev parameters.

Table 10-1

<i>Name</i>	<i>Device</i>
DW	RD50 Fixed 5-Megabyte Hard Disk
DZ	RX50 5¼-inch Diskette
TT	Console Terminal and Printer Port (TT1 for Console TT2 for Printer Port)

**10.6.5.2 Pseudo-Device Names** — A pseudo-device is a logical device that is redirected to another physical device unit. The pseudo-device provides device independence for standard naming conventions. The pseudo-devices in Table 10-2 are supported, as indicated.

Table 10-2

<i>Code</i>	<i>Device</i>
CL	Console Logger device, the hard copy output device (see LP).
LB	System library device.
LP	The printer port device.
SY	User default device.
TI	Pseudo-input terminal; TI1: is the terminal from which a task was requested.

### 10.6.6 The GLUN\$ Macro: Retrieving LUN Information

The Get LUN Information macro requests that information about a LUN-physical device unit association be returned in a 6-word buffer specified by the issuing task. Upon successful completion of the directive processing, the buffer contains the information listed in Table 10-3, as appropriate for the specific device. All three forms of the macro call may be used. It is issued from a MACRO-11 program in the following way:

```
GLUN$ lun,buf
```

lun     The logical unit number associated with the physical device unit for which information is requested.

buf     The 6-word buffer to which information is returned.

For example, to request information on the disk unit associated with LUN 8, issue the following call:

```
GLUN$C 8.,IOBUF
```

The example included below illustrates the use of the three forms of the GLUN\$ macro.

```

;
; DATA DEFINITIONS
;
GETLUN: GLUN$   6,DSKBUF       ; GENERATE DPB
      .
      .
      .
;
; EXECUTABLE SECTION
;
      DIR$     #GETLUN        ; EXECUTE DIRECTIVE
      .
      .
      .
      GLUN$C   6,DSKBUF       ; GENERATE DPB IN SEPARATE PROGRAM
      .                 ; SECTION, THEN GENERATE CODE TO
      .                 ; EXECUTE THE DIRECTIVE
      .
      GLUN$S   #6,#DSKBUF     ; GENERATE DPB ON STACK, THEN
      .                 ; EXECUTE DIRECTIVE

```



Table 10-3  
Get LUN Information

Numerical Offset			Symbolic Offset			Contents
Word	Byte	Bit	Word	Byte	Bit	
0			G.LUNA			Name of device associated with LUN (ASCII bytes)
1	0			G.LUNU		Unit number of associated device
	1			G.LUFB		Driver flag value. Returned as 200 <sub>8</sub> if the driver is resident, or as 0 if a loadable driver is not in the system
2			G.LUCW			First device characteristics word:
0			(U.CW1) <sup>1</sup>		(DV.REC)	Unit record-oriented device (for example, line printer) (1 = yes)
	1				(DV.CCL)	Carriage-control device (for example, line printer, terminal) (1 = yes)
	2				(DV.TTY)	Terminal device (1 = yes)
	3				(DV.DIR)	Directory device (for example, disk) (1 = yes)
					(DV.MSD)	Mass storage device (for example, disks) (1 = yes)
	8				(DV.EXT)	Device supports 22-bit direct addressing
	9				(DV.SWL)	Unit software write-locked (1 = yes)
	12				(DV.PSE)	Pseudo-device (1 = yes)
	14				(DV.F11)	Device mountable as a FILES-11 device (for example, disk (1 = yes)
	15				(DV.MNT)	Device mountable (logical OR of bits 13 and 14) (1 = yes)
3			G.LUCW+02			Second device characteristics word:
			(U.CW2)	(U2.xxx)		Device-specific information
4			G.LUCW+04			Third device characteristics word:
			(U.CW3)		(U3.xxx)	Device-specific information <sup>2</sup>
5			G.LUCW+06			Fourth device characteristics word:
			(U.CW4)			Default buffer size (for example, for disks, and line length for terminals).

1. For mass storage devices, such as disks, this is the number of blocks (maximum logical block number plus one).
2. The word and bit symbols shown in parentheses are symbols used in defining and referencing corresponding items in the device UCB.

### 10.6.7 The ASTX\$\$ Macro: Terminating AST Service

The AST Service Exit macro is used to terminate execution of an AST service routine. All forms of the macro are provided. However, the S-form is preferred because it requires less space and executes at least as fast as the ASTX\$ or ASTX\$C form of the macro. The macro is issued in the following way:

```
ASTX$$ [err]
```

err An optional argument that specifies the address of an error routine to which control branches if the directive is rejected.

On completion of the operation specified in this macro call, if another AST is queued and asynchronous system traps have not been disabled, then the next AST is immediately entered. Otherwise, the task's state before the AST was entered is restored (it is the AST service routine's responsibility to save and restore the registers it uses).

### 10.6.8 The WTSE\$ Macro: Waiting for an Event Flag

The Wait For Single Event Flag macro instructs the system to suspend execution of the issuing task until the event flag specified in the macro call is set. This macro is extremely useful in synchronizing activity on completion of an I/O operation. All three forms of the macro call may be used. It is issued as follows:

```
WTSE$ efn
```

efn The event flag number.

WTSE\$ causes the task to be blocked from execution until the specified event flag is set. Frequently, an efn parameter is also included in a QIO\$ macro call, and the event flag is set on completion of the I/O operation specified in that call.

The following example illustrates task blocking until the setting of the specified event flag occurs. This example also illustrates the use of the three forms of the macro call.

```

;
; DATA DEFINITIONS
;
WAIT:  WTSE$  5          ; GENERATE DPB
IOSB:  .BLKW  2          ; I/O STATUS BLOCK
.
.
.
;
; EXECUTABLE SECTION
;
ALUN$$ #14.,#'DW        ; ASSIGN LUN 14 TO UNIT ZERO
QIO$C  IO.ATT,14.,5     ; ATTACH DEVICE
DIR$   #WAIT           ; EXECUTE WAITFOR DIRECTIVE
.
.
.
QIO$$  #IO.RLB,#14.,#2,,#IOSB,,<#BUF,#80.>
.
.
.
.
WTSE$$ #2              ; WAIT FOR READ TO COMPLETE
.
.
.
QIO$C  IO.WLB,14.,3,,IOSB,,<BUF,80.>
.
.
.
.
WTSE$C 3              ; WAIT FOR WRITE TO COMPLETE
.
.
.
.
QIO$C  IO.DET,14.      ; DETACH DEVICE
.
.
.

```

## 10.7 STANDARD I/O FUNCTIONS

The number of I/O operations that can be specified by means of the QIO directive is large. A particular operation can be requested by including the appropriate function code as the first parameter of a QIO macro call. Certain functions are standard. These functions are almost totally device independent and can thus be requested for nearly every device described in this manual. Others are

device dependent and are specific to the operation of only one or two I/O devices. This section summarizes the function codes and characteristics of the following device-independent I/O operations:

- Attaching to an I/O device
- Detaching from an I/O device
- Cancelling I/O requests
- Reading a logical block
- Reading a virtual block
- Writing a logical block
- Writing a virtual block

For certain physical device units discussed in this manual, a standard I/O function may be described as being a NOP. This means that no operation is performed as a result of specifying the function, and an I/O status code of IS.SUC is returned in the I/O status block specified in the QIO macro call.

Note: In the following descriptions, the five QIO directive parameters lun, efn, pri, isb, and ast are represented by an ellipsis (...).

### 10.7.1 IO.ATT: Attaching to an I/O Device

The function code IO.ATT is specified by a user task when that task requires exclusive use of an I/O device. This function code is included as the first parameter of a QIO macro call in the following way:

```
QIO$C IO.ATT,...
```

Successful completion of an IO.ATT request causes the specified physical device unit to be dedicated for exclusive use by the issuing task. This enables the task to process input or output in an unbroken stream and is especially useful on sequential, non-file-oriented devices such as terminals. An attached physical device unit remains under control of the issuing task until it is explicitly detached by that task. To detach the device, the task can specify any LUN previously assigned to the attached device.

While a physical device unit is attached, the I/O driver for that unit dequeues only those I/O requests issued by the task that issued the attach. Thus, a request to attach a device unit already attached by another task will not be processed until the attachment is broken and no higher priority request exists for the unit. A LUN that is associated with an attached physical device unit may not be reassigned by means of an Assign LUN directive except when at least one LUN is still assigned to the attached device.

If the task that issued an attach function exits or is aborted before it issues a corresponding detach, the Executive automatically detaches the physical device unit.

### 10.7.2 IO.DET: Detaching from an I/O Device

The function code IO.DET is used to detach a physical device unit that has been previously attached by means of an IO.ATT request for exclusive use of the issuing task. This function code is included as the first parameter of a QIO macro call in the following way:

```
QIO$C IO.DET,...
```

The LUN specifications of both IO.ATT and IO.DET must be the same, as in the following example, which also illustrates the use of S- forms of several macro calls.

```
.MCALL  ALUN$$,QIO$$
ALUN$$  #14.,#'TT      ; ASSOCIATE TERMINAL WITH LUN 14.
.
.
.
QIO$$   #IO.ATT,#14.   ; ATTACH TERMINAL
.
.
.
LOOP:   QIO$$   #IO.RLB,#14.,... ; READ
.
.
.
QIO$$   #IO.DET,#14.   ; DETACH TERMINAL
```

### 10.7.3 IO.KIL: Canceling I/O Requests

The function IO.KIL is issued by a task to cancel all of that task's I/O requests for a particular physical device unit.

For I/O requests waiting for service—that is, in the I/O driver's queue—a status code of IE.ABO is returned in the I/O status block. An event flag is set, if specified. But any AST service routine that may have been specified is not initiated.

For I/O requests being processed by an I/O driver—other than the disk drivers—the IE.ABO status code is returned. Other status information (byte count, and so forth) is also returned in the I/O status block. An AST, if specified, is activated.

For disk I/O requests being processed when an IO.KIL is issued, the IO.KIL acts as a NOP. The request is allowed to complete. Because disks operate quickly, IO.KIL simply causes the return of IS.SUC in the I/O status block.

This function code is included as the first parameter of a QIO macro in the following way:

```
QIO$C IO.KIL,...
```

IO.KIL is useful in such special cases as canceling an I/O request on a physical device unit from which a response is overdue.

**10.7.4 IO.RLB: Reading a Logical Block**

The function code IO.RLB is specified by a task to read a block of data from the physical device unit specified in the macro call. This function code is included as the first parameter of a QIO macro in the following way:

```
QIO$C IO.RLB,...,<stadd,size,pn>
```

stadd The starting address of the data buffer.

size The data buffer size in bytes.

pn One to four optional parameters, used to specify such additional information as block numbers for certain devices.

**10.7.5 IO.RVB: Reading a Virtual Block**

The function code IO.RVB is used to read a virtual block of data from the physical device unit specified in the macro call. A “virtual” block indicates a relative block position within a file and is identical to a “logical” block for such sequential, record-oriented devices as terminals. For these sequential, record-oriented devices, IO.RVB is converted to IO.RLB before being issued.

Note: Any subfunction bits specified in the IO.RVB request (see Section 12.3.1) are stripped off in this conversion.

It is recommended that all tasks use virtual rather than logical reads. However, if a virtual read is issued for a file-structured device (disk), the user must ensure that a file is open on the specified physical device unit. This function code is included as the first parameter of a QIO macro call in the following way:

```
QIO$C IO.RVB,...,<stadd,size,pn>
```

stadd The starting address of the data buffer.

size The data buffer size in bytes.

pn One to four optional parameters, used to specify such additional information as block numbers for certain devices.

**10.7.6 IO.WLB: Writing a Logical Block**

The function code IO.WLB is specified by a task to write a block of data to the physical device unit specified in the macro call. This function code is included as the first parameter of a QIO macro call in the following way:

```
QIO$C IO.WLB,...,<stadd,size,pn>
```

stadd The starting address of the data buffer.

size The data buffer in bytes.

pn One to four optional parameters, used to specify such additional information as block numbers or format control characters for certain devices.

### 10.7.7 IO.WVB: Writing a Virtual Block

The function code IO.WVB is used to write a virtual block of data to a physical device unit. A "virtual" block indicates a block position relative to the start of a file. For sequential, record-oriented devices such as terminals and line printers, the function IO.WVB is converted to IO.WLB.

Note: Any subfunction bits specified in the IO.WVB request (see Section 12.3.1) are stripped off in this conversion.

It is recommended that all tasks use virtual rather than logical writes. However, if a virtual write is issued for a file-structured device (disk), the user must ensure that a file is open on the specified physical device unit. This function code is included as the first parameter of a QIO macro call in the following way:

```
QIO$C IO.WVB,...,<stadd,size,pn>
```

stadd The starting address of the data buffer.

size The data buffer size in bytes.

pn One to four optional parameters, used to specify such additional information as block numbers or format control characters for certain devices.

### 10.8 I/O COMPLETION

When an I/O request has been completed, either successfully or unsuccessfully, one or more actions may be taken by the Executive. Selection of return conditions depends on the parameters included in the QIO macro call. There are three major returns:

1. A significant event is declared on completion of an I/O operation. If an efn parameter was included in the I/O request, the corresponding event flag is set.
2. If an isb parameter was specified in the QIO macro call, a code identifying the type of success or failure is returned in the low-order byte of the first word of the I/O status block at the location represented by isb.

This status return code is of the form IS.xxx (success) or IE.xxx (error). For example, if the device accessed by the I/O request is not ready, a status code of IE.DNR is returned in isb. The section below (Return Codes) summarizes general codes returned by most of the drivers described in this manual.

If the isb parameter was omitted, the requesting task cannot determine whether the I/O request was successfully completed. A carry clear return from the directive itself simply means that the directive was accepted and the I/O request was queued, not that the actual input/output operation was successfully performed.

3. If an ast parameter was specified in the QIO macro call, a branch to the AST service routine that begins at the location identified by ast occurs on completion of the I/O operation. See Chapter 5 for a detailed description of AST service routines.

## 10.9 RETURN CODES

There are two kinds of status conditions recognized and handled by the system when they occur in I/O requests:

1. Directive conditions, which indicate the acceptance or rejection of the QIO directive itself
2. I/O status conditions, which indicate the success or failure of the I/O operation

Directive conditions relevant to I/O operations may indicate any of the following:

- Directive acceptance
- Invalid buffer specification
- Invalid efn parameter
- Invalid lun parameter
- Invalid DIC number or DPB size
- Unassigned LUN
- Insufficient memory

A code indicating the acceptance or rejection of a directive is returned to the Directive Status Word at symbolic location \$DSW. This location can be tested to determine the type of directive condition.

I/O conditions indicate the success or failure of the I/O operation specified in the QIO directive. I/O driver errors include such conditions as device not ready, privilege violation, file already open, or write-locked device. If an isb parameter is included in the QIO directive, identifying the address of a 2-word I/O status block, an I/O status code is returned in the low-order byte of the first word of this block on completion of the I/O operation. This code is a binary value corresponding to a symbolic name of the form IS.xxx or IE.xxx. The low-order byte of the word can be tested symbolically, by name, to determine the type of status return. The correspondence between global symbolic names and directive and I/O completion status codes is defined in the system object module library. Local symbolic definitions may also be obtained by the DRERR\$ and IOERR\$ macros.



Binary values of status codes always have the meanings indicated in Table 10-4.

Table 10-4  
Binary Status Codes

<i>Code</i>	<i>Meaning</i>
Positive (greater than 0)	Successful completion
0	Operation still pending
Negative	Unsuccessful completion

A pending operation means that the I/O request is still in the queue of requests for the respective driver, or the driver has not yet completely serviced the request.

### 10.9.1 Directive Conditions

Table 10-5 summarizes the directive conditions that may be encountered in QIO directives. The acceptance condition is first, followed by error codes indicating various reasons for rejection, in alphabetical order. (See Appendix A for a summary of error codes.)

Table 10-5  
Directive Conditions

<i>Code</i>	<i>Reason</i>
IS.SUC	<i>Directive accepted</i> The first six parameters of the QIO directive were valid, and sufficient dynamic memory was available to allocate an I/O packet. The directive is accepted.
IE.ADP	<i>Invalid address</i> The I/O status block or the QIO DPB was outside of the issuing task's address space or was not aligned on a word boundary.
IE.HWR	<i>Device handler not resident</i> The driver for the requested device was not loaded in memory.
IE.IEF	<i>Invalid event flag number</i> The efn specification in a QIO directive was less than 0 or greater than 96.
IE.ILU	<i>Invalid logical unit number</i> The lun specification in a QIO directive was invalid for the issuing task. For example, there were only five logical unit numbers associated with the task, and the value specified for lun was greater than 5.
IE.PRI	<i>Privilege violation</i> The user does not have the required privilege for the requested operation.

Table 10-5 (Cont.)

<i>Code</i>	<i>Reason</i>
IE.SDP	<p><i>Invalid DIC number or DPB size</i></p> <p>The directive identification code (DIC) or the size of the Directive Parameter Block (DPB) was incorrect; the legal range for a DIC is from 1 through 127, and all DIC values must be odd. Each individual directive requires a DPB of a certain size. If the size is not correct for the particular directive, this code is returned. The size of the QIO DPB is always 12 words.</p>
IE.ULN	<p><i>Unassigned LUN</i></p> <p>The logical unit number in the QIO directive was not associated with a physical device unit. The user may recover from this error by issuing a valid Assign LUN directive and then reissuing the rejected directive.</p>
IE.UPN	<p><i>Insufficient dynamic memory</i></p> <p>There was not enough dynamic memory to allocate an I/O packet for the I/O request. The user can try again later by blocking the task with a Waitfor Significant Event directive. Note that Waitfor Significant Event is the only effective way for the issuing task to block its execution, since other directives that could be used for this purpose themselves require dynamic memory for their execution (for example, Mark Time).</p>

### 10.9.2 I/O Status Conditions

The following list summarizes status codes that may be returned in the I/O status block specified in the QIO directive on completion of the I/O request. The I/O status block is a 2-word block with the following format:

- The low-order byte of the first word receives a status code of the form IS.xxx or IE.xxx on completion of the I/O operation.
- The high-order byte of the first word is usually device dependent. In cases where the user might find information in this byte helpful, this manual identifies that information.
- The second word contains the number of bytes transferred or processed if the operation is successful and involves reading or writing.

If the *isb* parameter of the QIO directive is omitted, this information is not returned.

The following illustrates a sample 2-word I/O status block on completion of a terminal read operation:

```

      1  0  Byte
Word 0  0  -10
      1  Number of bytes read

```

where -10 is the status code for IE.EOF (end of file). If this code is returned, it indicates that input was terminated by typing CTRL/Z, which is the end-of-file termination sequence on a terminal.

To test for a particular error condition, the user generally compares the low-order byte of the first word of the I/O status block with a symbolic value, as in the following:

CMPB #IE.DNR,IOSB

However, to test for certain types of successful completion of the I/O operation, the entire word value must be compared. For example, if a carriage return terminated a line of input from the terminal, a successful completion code of IS.CR is returned in the I/O status block. If an Escape character was the terminator, a code of IS.ESC is returned. To check for these codes, the user should first test the low-order byte of the first word of the block for IS.SUC and then test the full word for IS.CC, IS.CR, IS.ESC, or IS.ESQ.

Note that both of the following comparisons will test as equal since the low-order byte in both cases is +1.

CMP #IS.CR,IOSB

CMPB #IS.SUC,IOSB

In the case of a successful completion where the carriage return is the terminal indicator (IS.CR), the following illustrates the status block:

	1	0	Byte
Word 0	15	+1	
	1 Number of bytes read (excluding the CR)		

where 15 is the octal code for carriage return and +1 is the status code for successful completion.

The codes described in Table 10-6 are general status codes that apply to the majority of devices presented in Chapters 11 and 12. Error codes specific to only one or two drivers are described only in relation to the devices for which they are returned. The list below describes successful and pending codes first, then error codes in alphabetical order.

Table 10-6  
I/O Status Conditions

<i>Code</i>	<i>Meaning</i>
IS.SUC	<i>Successful completion</i> The I/O operation specified in the QIO directive was completed successfully. The second word of the I/O status block can be examined to determine the number of bytes processed, if the operation involved reading or writing.
IS.PND	<i>I/O request pending</i> The I/O operation specified in the QIO directive has not yet been executed. The I/O status block is filled with 0s.

Table 10-6 (Cont.)

<i>Code</i>	<i>Meaning</i>
IE.ABO	<i>Operation aborted</i> The specified I/O operation was cancelled with IO.KIL while in progress or while still in the I/O queue.
IE.ALN	<i>File already open</i> The task attempted to open a file on the physical device unit associated with the specified LUN, but a file has already been opened by the issuing task on that LUN.
IE.BAD	<i>Bad parameter</i> An illegal specification was supplied for one or more of the device-dependent QIO parameters (words 6-11). For example, a bad channel number or gain code was specified in an analog-to-digital converter I/O operation.
IE.BBE	<i>Bad block on device</i> One or more bad blocks were found by executing the BAD utility. Data cannot be written on bad blocks.
IE.BLK	<i>Illegal block number</i> An illegal block number was specified for a file-structured physical device unit.
IE.BYT	<i>Byte-aligned buffer specified</i> Byte alignment was specified for a buffer, but only word (or double-word) alignment is legal for the physical device unit. For example, a disk function requiring word alignment was requested, but the buffer was aligned on a byte boundary.
IE.DAA	<i>Device already attached</i> The physical device unit specified in an IO.ATT function was already attached to the issuing task. This code indicates that the issuing task has already attached the desired physical device unit, not that the unit was attached by another task.
IE.DNA	<i>Device not attached</i> The physical device unit specified in an IO.DET function was not attached to the issuing task. This code has no bearing on the attachment status with respect to other tasks.
IE.DNR	<i>Device not ready</i> The physical device unit specified in the QIO directive was not ready to perform the desired I/O operation. This code is often returned as the result of an interrupt time-out; that is, a "reasonable" amount of time has passed, and the physical device unit has not responded.
IE.EOF	<i>End-of-file encountered</i> An end-of-file mark, record, or control character was recognized on the input device.
IE.FHE	<i>Fatal hardware error</i> Controller is physically unable to reach the location where input/output is to be performed on the device. The operation cannot be completed.
IE.IFC	<i>Illegal function</i> A function code was specified in an I/O request that was illegal for the specified physical device unit. This code is returned if the task attempts to execute an illegal function or if, for example, a read function is requested on an output-only device, such as the line printer.

Table 10-6 (Cont.)

<i>Code</i>	<i>Meaning</i>
IE.NLN	<i>File not open</i> The task attempted to close a file on the physical device unit associated with the specified LUN, but no file was currently open on that LUN.
IE.NOD	<i>Insufficient buffer space</i> Dynamic storage space has been depleted, and there was insufficient buffer space available to allocate a secondary control block. For example, if a task attempts to open a file, buffer space for the window and file control block must be supplied by the Executive. This code is returned when there is not enough space for such an operation.
IE.OFL	<i>Device off line</i> The physical device unit associated with the LUN specified in the QIO directive was not on line. When the system was booted, a device check indicated that this physical device unit was not in the configuration.
IE.OVR	<i>Illegal read overlay request</i> A read overlay was requested and the physical device unit specified in the QIO directive was not the physical device unit from which the task was installed. The read overlay function can only be executed on the physical device unit from which the task image containing the overlays was installed.
IE.PRI	<i>Privilege violation</i> The task that issued a request was not privileged to execute that request.
IE.SPC	<i>Illegal address space</i> The buffer requested for a read or write request was partially or totally outside the address space of the issuing task. Alternately, a byte count of 0 was specified.
IE.VER	<i>Unrecoverable error</i> After the system's standard number of retries have been attempted upon encountering an error, the operation still could not be completed.
IE.WCK	<i>Write check error</i> An error was detected during the check (read) following a write operation.
IE.WLK	<i>Write-locked device</i> The task attempted to write on a write-locked physical device unit.



# CHAPTER 11

## DISK DRIVERS

---

The system's disk drivers support the disks summarized in Table 11-1. Subsequent sections describe these devices in greater detail.

All of the disks described in this chapter are accessed in essentially the same manner. Disks and other file-structured media are divided logically into a series of 256-word blocks.

Table 11-1  
Standard Disk Devices

<i>Controller/ Drive</i>	<i>RPM</i>	<i>Secs</i>	<i>Heads</i>	<i>Cylinders</i>	<i>Bytes/ Drive</i>	<i>Decimal Blocks</i>
RX50	300	10	2	80/diskette	819,200	800
RD50	3600	16	4	153/surface	5MB	9727

### 11.1 RX50 DESCRIPTION

The RX50 (diskette) subsystem consists of a 5.25-inch dual flexible diskette drive and a separate single-board controller module. The module enables a data processing system to store or retrieve information from any location on one side of each front-loadable diskette.

### 11.2 RD50 DESCRIPTION

The RD50 (hard disk) is a random-access, rotating memory device. It stores data in fixed-length blocks on 130mm rigid disk media. Winchester technology uses moving head, noncontact recording. The drive contains a storage media in a fixed configuration which cannot be removed.

### 11.3 GET LUN INFORMATION MACRO

Word 2 of the buffer filled by the Get LUN Information system directive (the first characteristics word) contains the information listed in Table 11-2 for disks. A bit setting of 1 indicates that the described characteristic is true for disks.

Table 11-2

<i>Bit</i>	<i>Setting</i>	<i>Meaning</i>
0	0	Record-oriented device
1	0	Carriage-control device
2	0	Terminal device
3	1	File-structured device
4	0	Single-directory device
5	0	Sequential device
6	1	Mass storage device
7	X	User-mode diagnostics supported (device dependent)
8	X	Device supports 22-bit direct addressing
9	0	Unit software write-locked
10	0	Input spooled device
11	0	Output spooled device
12	0	Pseudo-device
13	0	Device mountable as a communications channel
14	1	Device mountable as a Files-11 volume
15	1	Device mountable

Words 3 and 4 of the buffer contain the maximum logical block number. Note that the high byte of U.CW2 is undefined. The user should clear the high byte in the buffer before using the block number. Word 5 indicates the default buffer size, which is 512 bytes for all disks.



## 11.4 QIO MACRO

This section summarizes the standard and the device-specific QIO functions for disk drivers.

### 11.4.1 Standard QIO Functions

Table 11-3 lists the standard functions of the QIO macro that are valid for disks.

Table 11-3  
Standard QIO Functions for Disks

<i>Format</i>	<i>Function</i>
QIO\$C IO.ATT,,	Attach device <sup>1</sup>
QIO\$C IO.DET,,	Detach device
QIO\$C IO.KIL,,	Kill I/O <sup>2</sup>
QIO\$C IO.RLB,,<stadd,size,,blkh,blkI>	READ logical block
QIO\$C IO.RVB,,<stadd,size,,blkh,blkI>	READ virtual block
QIO\$C IO.WLB,,<stadd,size,,blkh,blkI>	WRITE logical block
QIO\$C IO.WLC,,<stadd,size,,blkh,blkI>	WRITE logical block followed by write check
QIO\$C IO.WVB,,<stadd,size,,blkh,blkI>	WRITE virtual block

**stadd**            The starting address of the data buffer (must be on a word boundary).

**size**             The data buffer size in bytes (must be even or greater than 0).

**blkh/blkI**        Block high and block low, combining to form a double-precision number that indicates the actual logical/virtual block address on the disk where the transfer starts; blkh represents the high 8 bits of the address, and blkI the low 16 bits.

IO.RVB and IO.WVB are associated with file operations. For these functions to be executed, a file must be open on the specified LUN if the volume associated with the LUN is mounted. Otherwise, the virtual I/O request is converted to a logical I/O request using the specified block numbers.

Note: When writing a new file using QIOs, the task must explicitly issue .EXTND File Control System library routine calls as necessary to reserve enough blocks for the file, or the file must be initially created with enough blocks allocated for the file. In addition, the task must put an appropriate value in the FDB for the end-of-file block number (F.EFBK) before closing the file.

1. Only volumes mounted foreign may be attached. Any other attempt to attach a mounted volume will result in an IE.PRI status being returned in the I/O status doubleword.
2. In-progress disk operations are allowed to complete when IO.KIL is received, because they take such a short time. I/O requests that are queued when IO.KIL is received are killed immediately. An IE.ABO status is returned in the I/O status doubleword.

Each disk driver supports the subfunction bit IQ.X: inhibit retry attempts for error recovery. The subfunction bit is used by ORing it into the desired QIO; for example:

```
QIO$C IO.WLB!IQ.X,...,<stadd,size,,blkh,blk!>
```

The IQ.X subfunction permits user-specified retry algorithms for applications in which data reliability must be high.

## 11.5 STATUS RETURNS

The error and status conditions listed in Table 11-4 are returned by the disk drivers described in this chapter.

Table 11-4  
Disk Status Returns

<i>Code</i>	<i>Reason</i>
IS.SUC	<i>Successful completion</i> The operation specified in the QIO directive was completed successfully. The second word of the I/O status block can be examined to determine the number of bytes processed, if the operation involved reading or writing.
IS.PND	<i>I/O request pending</i> The operation specified in the QIO directive has not yet been executed. The I/O status block is filled with 0s.
IE.ABO	<i>Request aborted</i> An I/O request was queued (not yet acted upon by the driver) when an IO.KIL was issued.
IE.ALN	<i>File already open</i> The task attempted to open a file on the physical device unit associated with specified LUN, but a file has already been opened by the issuing task on that LUN.
IE.BLK	<i>Illegal block number</i> An illegal logical block number was specified.
IE.BBE	<i>Bad block error</i> The disk sector (block) being read was marked as a bad block in the header word.
IE.BYT	<i>Byte-aligned buffer specified</i> Byte alignment was specified for a buffer, but only word alignment is legal for disk. Alternatively, the length of a buffer is not an appropriate number of bytes.
IE.DNR	<i>Device not ready</i> The physical device unit specified in the QIO directive was not ready to perform the desired I/O operation.
IE.FHE	<i>Fatal hardware error</i> The controller is physically unable to reach the location where input/output operation is to be performed. The operation cannot be completed.

Table 11-4 (Cont.)

<i>Code</i>	<i>Reason</i>
IE.IFC	<i>Illegal function</i> A function code was specified in an I/O request that is illegal for disks.
IE.MII	<i>Media inserted incorrectly</i> The controller has detected that the media (such as a floppy diskette) was not inserted correctly. To correct the problem, reinsert the media properly.
IE.NLN	<i>File not open</i> The task attempted to close a file on the physical device unit associated with the specified LUN, but no file was currently open on that LUN.
IE.NOD	<i>Insufficient buffer space</i> Dynamic storage space has been depleted, and there was insufficient buffer space available to allocate a secondary control block. For example, if a task attempts to open a file, buffer space for the window and file control block must be supplied by the Executive. This code is returned when there is not enough space for this operation.
IE.OFL	<i>Device off line</i> The physical device unit associated with the LUN specified in the QIO directive was not on line. When the system was booted, a device check indicated that this physical device unit was not in the configuration.
IE.OVR	<i>Illegal read overlay request</i> A read overlay was requested, and the physical device unit specified in the QIO directive was not the physical device unit from which the task was installed. The read overlay function can only be executed on the physical device unit from which the task image containing the overlays was installed.
IE.PRI	<i>Privilege violation</i> The task that issued the request was not privileged to execute that request. For disk, this code is returned if a nonprivileged task attempts to read or write a mounted volume directly (that is, using IO.RLB or IO.WLB). Also, this code is returned if any task attempts to attach a mounted volume.
IE.SPC	<i>Illegal address space</i> The buffer specified for a read or write request was partially or totally outside the address space of the issuing task. Alternately, a byte count of 0 was specified.
IE.VER	<i>Unrecoverable error</i> After the system's standard number of retries has been attempted upon encountering an error, the operation still could not be completed. For disk, unrecoverable errors are usually parity errors.
IE.WCK	<i>Write check error</i> An error was detected during the write check portion of an operation.
IE.WLK	<i>Write-locked device</i> The task attempted to write on a disk that was write-locked.

When a disk I/O error condition is detected, an error is usually not returned immediately. Instead, the system attempts to recover from most errors by retrying the function as many as eight times. Unrecoverable errors are generally parity, timing, or other errors caused by a hardware malfunction.



# CHAPTER 12

## THE TERMINAL DRIVER

---

### 12.1 INTRODUCTION

The system supports a single full-duplex terminal driver which includes the following features:

- Full-duplex operation
- Type-ahead buffering
- Eight-bit characters
- Transparent read and write
- Formatted read and write
- Read after prompt
- Read with no echo
- Read with special terminator
- Optional time-out on solicited input
- Device-independent cursor control

## 12.2 GET LUN INFORMATION MACRO

Word 2 of the buffer filled by the Get LUN Information system directive (the first characteristics word) contains the information noted in Table 12-1 for terminals. A setting of 1 indicates that the described characteristic is true for terminals.

Table 12-1  
Buffer Get LUN Information

<i>Bit</i>	<i>Setting</i>	<i>Meaning</i>
0	1	Record-oriented device
1	1	Carriage-control device
2	1	Terminal device
3	0	File-structured device
4	0	Single-directory device
5	0	Sequential device
6	0	Mass storage device
7	0	User-mode diagnostics supported
8	0	Device supports 22-bit direct addressing
9	0	Unit software write-locked
12	0	Pseudo device
13	0	Device mountable as a communications channel
14	0	Device mountable as a Files-11 volume
15	0	Device mountable

Words 3 and 4 of the buffer are undefined. Word 5 indicates the default buffer size (the width of the terminal carriage or display screen).

## 12.3 QIO MACRO

Table 12-2 lists the standard and device-specific functions of the QIO Macro that are valid for terminals.

Table 12-2  
Standard and Device-Specific QIO Functions for Terminals

<i>Format</i>	<i>Function</i>
<i>Standard Functions:</i>	
QIO\$C IO.ATT,...	Attach device
QIO\$C IO.DET,...	Detach device
QIO\$C IO.KIL,...	Cancel I/O requests
QIO\$C IO.RLB,....,<stadd,size[,tmo]>	READ logical block (read typed input into buffer).
QIO\$C IO.RVB,....,<stadd,size[,tmo]>	READ virtual block (read typed input into buffer).
QIO\$C IO.WLB,....,<stadd,size,vfc>	WRITE logical block (print buffer contents).
QIO\$C IO.WVB,....,<stadd,size,vfc>	WRITE virtual block (print buffer contents).
<i>Device-Specific Functions:</i>	
QIO\$C IO.ATA,....,<ast, [parameter2][,ast2]>	ATTACH device, specify unsolicited-input- character AST
QIO\$C IO.CCO,....,<stadd,size,vfc>	CANCEL CTRL/O (if in effect), then write logical block
QIO\$C SF.GMC,....,<stadd,size>	GET multiple characteristics
QIO\$C IO.GTS,....,<stadd,size>	GET terminal support
QIO\$C IO.RAL,....,<stadd,size[,tmo]>	READ logical block, pass all bits
QIO\$C IO.RNE,....,<stadd,size[,tmo]>	READ logical block, do not echo
QIO\$C IO.RPR,....,<stadd,size, [tmo],pradd,prsize,vfc>	READ logical block after prompt
QIO\$C IO.RST,....,<stadd,size[,tmo]>	READ logical block ended by special terminators
QIO\$C IO.RTT,....,<stadd,size, [tmo],table>	READ logical block ended by specified special terminator
QIO\$C SF.SMC,....,<stadd,size>	SET multiple characteristics
QIO\$C IO.WAL,....,<stadd,size,vfc>	WRITE logical block, pass all bits
QIO\$C IO.WSD,....,<stadd,size,,type>	WRITE special data
QIO\$C IO.RSD,....,<stadd,size,tmo,type>	READ special data

ast	The entry point for an unsolicited- input-character AST.
parameter 2	A number that can be used to identify this terminal as the input source upon entry to an unsolicited character AST routine.
ast2	The entry point for an INTERRUPT/DO sequence AST. (See Section 12.5.2)
pradd	The starting address of the byte buffer where the prompt is stored.
prsize	The size of the pradd prompt buffer in bytes. The specified size must be greater than 0 and less than or equal to 8128. The buffer must be within the task's address space.
size	The size of the stadd data buffer in bytes. The specified size must be greater than 0 and less than or equal to 8128. The buffer must be within the task's address space. For SF.GMC, IO.GTS, and SF.SMC functions, size must be an even value.
stadd	The starting address of the data buffer. The address must be word aligned for SF.GMC, IO.GTS, and SF.SMC, IO.RSD, IO.WSD ; otherwise, stadd may be on a byte boundary.
table	The address of the 16-word special terminator table.
tmo	An optional time-out count in 10-second intervals. If 0 is specified, no time-out can occur. Time-out is the maximum time allowed between two input characters before the read is aborted.
type	The data type of the buffer contents.
vfc	A character for vertical format control from Table 12-12 (Vertical Format Control Characters).

### 12.3.1 Subfunction Bits

Most device-specified functions supported by terminal drivers described in this section are selected using "subfunction bits." One or more functions can be selected by ORing their relative bits in a QIO function. Table 12-4 contains a listing of QIO functions and relative subfunction bits that can be issued.



Each subfunction bit and subfunction selected when it is included in a QIO function is listed in Table 12-3.

Table 12-3

<i>Symbolic Name</i>	<i>Subfunction</i>
TF.AST	Unsolicited-input-character AST
TF.BIN	Binary prompt
TF.CCO	Cancel CTRL/O
TF.ESQ	Recognize escape sequences
TF.NOT	Unsolicited input AST notification; unsolicited characters are stored in the type-ahead buffer until they are read by the task
TF.RAL	Read all bits
TF.RCU	Restore cursor position
TF.RNE	Read with no echo
TF.RST	Read with special terminators
TF.TMO	Read with time-out
TF.WAL	Write all bits
TF.XCC	Sends an INTERRUPT/DO sequence to the P/OS Dispatcher.

Table 12-4 lists subfunction bits that can be ORed with QIO functions. Additional details for using subfunction bits are included in Section 12.3.2.

If a task invokes a subfunction bit that is not supported on the system, the subfunction bit is ignored, but the QIO request is not rejected.

The following example is a QIO request using more than one subfunction bit: a nonechoed (TF.RNE) read, terminated by a special terminator character (TF.RST) and preceded by a prompt.

```
QIO$C IO.RPR!TF.RNE!TF.RST,...,<stadd,size,,pradd,prsize,vfc>
```

### 12.3.2 Device-Specific QIO Functions

All functions except SF.GMC, IO.RPR, SF.SMC, IO.RTT, and IO.GTS can be issued by ORing a particular subfunction bit with another QIO function. These subfunction bits are specified in the following descriptions; subfunction bits are described in general in Section 12.3.1.

In addition to the device-specific QIO functions, this section also describes the use of subfunction bits TF.ESQ and TF.BIN.

Table 12-4  
Summary of Subfunction Bits

<i>Allowed Subfunction Bits</i>													
<i>Function</i>	<i>Equivalent Subfunctions</i>	<i>TF.AST</i>	<i>TF.BIN</i>	<i>TF.CCO</i>	<i>TF.ESQ</i>	<i>TF.NOT</i>	<i>TF.RAL</i>	<i>TF.RCU</i>	<i>TF.RNE</i>	<i>TF.RST</i>	<i>TF.TMO</i>	<i>TF.WAL</i>	<i>TF.XCC</i>
<i>Standard Functions</i>													
IO.ATT		X			X								
IO.DET													
IO.KIL													
IO.RLB							1	X	1	X			
IO.RVB							2			2	2		
IO.WLB				X				X					
IO.WVB				2				2				2	
<i>Device-Specific Functions</i>													
IO.ATA	IO.ATT!TF.AST				X	X							X
IO.CCO	IO.WLB!TF.CCO											3	
SF.GMC													
IO.GTS													
IO.RAL	IO.RLB!TF.RAL								X	1	X		
IO.RNE	IO.RLB!TF.RNE						1			1	X		
IO.RPR			X				1		X	1	X		
IO.RST	IO.RLB!TF.RST						1		X		X		
IO.RTT						1		X		X			X
SF.SMC													
IO.WAL	IO.WLB!TF.WAL			3				3					
IO.WBT	IO.WLB!TF.WBT			X				X				3	
IO.WSD													
IO.RSD											X		

## Notes:

- Exercise great care when using Read All and Read with Special Terminators together. Obscure problems can result.
- These subfunctions are allowed but are not effective. They are stripped off when the read or write virtual operation is converted to a read or write logical operation.
- During a write-pass-all operation (IO.WAL or IO.WLB!TF.WAL) the terminal driver outputs characters without interpretation; it does not keep track of cursor position.

**12.3.2.1 IO.ATA** — IO.ATA is a variation of the Attach function. The use of this function is eased by the addition of TF.NOT and TF.XCC subfunction bits, described later in this section. IO.ATA specifies asynchronous system traps (ASTs) to process unsolicited input characters. When called as follows:

```
QIO$C IO.ATA,....,<[AST],[PARAMETER2],[AST2]>
```

Note: A minimum of one AST parameter (ast or ast2) is required.

This function attaches the terminal and identifies "ast" and "ast2" as entry points for an unsolicited-input-character AST. Control passes to ast whenever an unsolicited character (other than CTRL/Q, CTRL/S, CTRL/X, or CTRL/O) is input. If the ast2 parameter is specified, an INTERRUPT/DO sequence results in the specified AST being entered in that parameter. If ast2 is not specified, an INTERRUPT/DO sequence results in the specified AST being entered in the ast parameter.

Unless the TF.XCC subfunction is specified, the INTERRUPT/DO sequence is trapped by the task and does not reach the P/OS Dispatcher. Thus, any task that uses IO.ATA without the TF.XCC subfunction should recognize some input sequence as a request to terminate; otherwise, the P/OS Dispatcher cannot be invoked to abort the task in case of difficulty.

Note that either ast2 or TF.XCC can be used, but not both in the same QIO request. If both are specified in the request, an IE.SPC error is returned.

Upon entry to the AST routines, the unsolicited character and parameter 2 are in the top word on the stack, as shown below. That word must be removed from the stack before exiting the AST.

SP+10	Event flag mask word
SP+06	PS of task prior to AST
SP+04	PC of task prior to AST
SP+02	Task's directive status word
SP+00	Unsolicited character in low byte; parameter 2, in the high byte, is a user-specified value that can be used to identify individual terminals in a multiterminal environment

The processing of unsolicited input ASTs is eased through the use of TF.NOT and TF.XCC subfunction bits. When TF.XCC is included in the IO.ATA function, all characters (except INTERRUPT/DO sequences) are handled in the manner previously described. INTERRUPT/DO sequences cause the P/OS Dispatcher to abort the application.

When unsolicited terminal input (except an INTERRUPT/DO sequence) is received by the terminal driver and the TF.NOT subfunction is used, the resulting AST serves only as notification of unsolicited terminal input; the terminal driver does not pass the character to the task. Upon entry to the AST service routine, the high byte of the first word on the stack identifies the terminal causing the AST (parameter 2). After the AST has been effected, the AST becomes "dis-

armed” until a read request is issued by the task. If multiple characters are received before the read request is issued, they are stored in the type-ahead buffer. Once the read request is received, the contents of the type-ahead buffer, including the character causing the AST, is returned to the task; the AST is then “armed” again for new unsolicited input characters. Thus, using the TF.NOT subfunction allows a task to monitor more than one terminal for unsolicited input without the need to continuously read each terminal for possible unsolicited input.

See Chapter 5 for further details on ASTs.

IO.ATA is equivalent to IO.ATT ORed with the subfunction bit TF.AST.

**12.3.2.2 IO.ATT!TF.ESQ** — The task issuing this function attaches a terminal and notifies the driver that it recognizes escape sequences input from that terminal. Escape sequences are recognized only for solicited input. (See Section 12.6 for a discussion of escape sequences.)

If the terminal has not been declared capable of generating escape sequences, IO.ATT!TF.ESQ has no effect other than attaching the terminal. No escape sequences are returned to the task because any ESC sent by the terminal acts as a line terminator. The SF.SMC function is used to declare the terminal capable of generating escape sequences (see Table 12-5 Driver-Terminal Characteristics for SF.GMC and SF.SMC Functions).

**12.3.2.3 IO.CCO** — This write function directs the driver to write to the terminal regardless of a CTRL/O condition that may be in effect. If CTRL/O is in effect, it is cancelled before the write is done.

IO.CCO is equivalent to IO.WLB!TF.CCO.

**12.3.2.4 SF.GMC** — The Get Multiple Characteristics function returns terminal characteristics information, as follows:

QIO\$C SF.GMC,...,<stadd,size>

stadd	The starting address of a data buffer of length “size” bytes. Each word in the buffer has the form
	.BYTE characteristic-name
	.BYTE 0
characteristic-name	One of the bit names given in Table 12-5. The value returned in the high byte of each byte-pair is 1 if the characteristic is true for the terminal and 0 if it is not true.

Table 12-5  
Driver-Terminal Characteristics for SF.GMC and SF.SMC Functions

<i>Bit Name</i>	<i>Octal Value</i>	<i>Meaning (if asserted)</i>
TC.ACR	24	Wrap-around mode
TC.BIN	65	Binary input mode (read-pass-all) no characters are interpreted as control characters.
TC.CTS	72	Suspend output to terminal 0 = resume 1 = suspend
TC.ESQ	35	Input escape sequence recognition
TC.FDX	64	Full-duplex mode
TC.HFF	17	Hardware form-feed capability (If 0, form-feeds are simulated using TC.LPP.)
TC.HFL	13	Number of fill characters to insert after a RETURN (0-7=x)
TC.HHT	21	Horizontal tab capability (if 0, horizontal tabs are simulated using spaces.)
TC.LPP	2	Page length (1-255.=x)
TC.NEC	47	Echo suppressed
TC.SCP	12	Terminal is a scope (CRT)
TC.SMR	25	Upper-case conversion disabled
TC.TBF	71	Type-ahead buffer count (read), or flush (write)
TC.TTP	10	Terminal type (=0-255.=x)
TC.VFL	14	Send four fill characters after line feed
TC.WID3	1	Page width (=1-255.=x)

For the TC.TTP characteristic (terminal type), one of the values shown in Table 12-6 is returned in the high byte.

The TC.TTP characteristic, when read by the terminal driver, sets implicit values for terminal characteristics TC.LPP, TC.WID, TC.HFF, TC.HHT, TC.VFL, and TC.SCP as shown in Table 12-6. These values can be changed (overridden) by subsequent Set Multiple Characteristics requests. In addition, TC.TTP is used by the terminal driver to determine cursor positioning commands, as appropriate.

Table 12-6  
TC.TTP (Terminal Type) Values Set by SF.SMC and Returned by SF.GMC

<i>Implicit Characteristics<sup>1</sup></i>								
<i>Symbolic</i>	<i>Terminal Type</i>	<i>TC.LPP</i>	<i>TC.WID</i>	<i>TC.HFF</i>	<i>TC.HHT</i>	<i>TC.HFL</i>	<i>TC.VFL</i>	<i>TC.SCP</i>
T.UNK0	Unknown							
T.V100	VT100	24	80		1			1
T.L120	LA120	66	132	1				
T.LA12	LA12	66	132	1	1			
T.L100	LA100	66	132	1	1			
T.V101	VT101	24	80		1			1
T.V102	VT102	24	80		1			1
T.V105	VT105	24	80		1			1
T.V125	VT125	24	80		1			1
T.LQP2	LQP02	66	80	1	1			
T.LA50	LA50	66	80	1	1			
T.BMP1	PC300 <sup>2</sup>	24	80					1

The TC.CTS characteristic returns the present suspend (CTRL/S), resume (CTRL/Q), or suppress (CTRL/O) state set via the SF.SMC function. Values returned are as follows:

Table 12-7

<i>Value Returned</i>	<i>State</i>
0	Resume (CTRL/Q)
1	Suspend (CTRL/S)
2	Suppress (CTRL/O)
3	Both suppress and suspend

When a value of 0 is used with the SF.SMC function, the suspend state is cleared; a value of 1 selects the suspend state.

Note: If you stop output to the terminal screen by pressing the Hold Screen key on a PC300, TC.CTS does not indicate that output has stopped. In addition, if you stop output to the terminal screen by pressing the NO SCROLL key on a VT100 series terminal or the Hold Screen key on a PC300 series terminal, output cannot be resumed with TC.CTS.

The TC.TBF characteristic returns the number of unprocessed characters in the type-ahead buffer for the specified terminal. This allows tasks to determine if any characters were typed that did not require AST processing. In addition, the value returned can be used to read the exact number of characters typed, rather than a typical value of 80<sub>10</sub> or 132<sub>10</sub> characters for the terminal.

1. Implicit characteristics are shown as supported by the driver. Values not shown are not automatically set by the driver. An "unknown" terminal type has no implicit characteristics.
2. The PC300 Series Bitmap Display is the default terminal for the Professional.

**Notes**

1. The maximum capacity of the type-ahead buffer is 36 characters.
2. Using TC.TBF in an SF.SMC function flushes the type-ahead buffer.

**12.3.2.5 IO.GTS** —This function is a Get Terminal Support request that returns information to a 4-word buffer specifying which features are part of the terminal driver. Only two of these words are currently defined.

The various symbols used by the IO.GTS, SF.GMC, and SF.SMC functions are defined in a system module, TTSYM. These symbols include:

F1.xxx and F2.xxx (Table 12-8)

T.xxxx (Table 12-6)

TC.xxx (Table 12-5)

The SE.xxx error returns described in Table 12-9.

These symbols may be defined locally within a code module by using:

```
.MCALL  TTSYM$
.
.
.
TTSYM$
```

Symbols that are not defined locally are automatically defined by the Task Builder.

Table 12-8  
Information Returned by Get Terminal Support (IO.GTS) QIO

<i>Mnemonic</i>	<i>Meaning When Set to 1</i>
Word 0 of Buffer:	
F1.ACR	Automatic CR/LF on long lines
F1.BUF	Checkpointing during terminal input
F1.UIA	Unsolicited-input-character AST
F1.CCO	Cancel CTRL/O before writing
F1.ESQ	Recognize escape sequences in solicited input
F1.LWC	Lower- to uppercase conversion
F1.RNE	Read with no echo
F1.RPR	Read after prompting
F1.RST	Read with special terminators
F1.RUB	CRT rubout
F1.TRW	Read all and write all
F1.UTB	Input characters buffered in task's address space
F1.VBF	Variable-length terminal buffers

Table 12-8 (Cont.)

<i>Mnemonic</i>	<i>Meaning When Set to 1</i>
Word 1 of Buffer:	
F2.SCH	Set characteristics QIO (SF.SMC)
F2.GCH	Get characteristics QIO (SF.GMC)
F2.SFF	Formfeed can be simulated
F2.CUP	Cursor positioning
F2.FDX	Full Duplex Terminal Driver

**12.3.2.6 IO.RAL** — The Read All function causes the driver to pass all bits to the requesting task. The driver does not intercept control characters or mask out the high-order bit. For example, CTRL/Q, CTRL/S, CTRL/O, and CTRL/Z and INTERRUPT/DO sequences are passed to the program and are not interpreted by the driver.

Note: IO.RAL echoes the characters that are read. To read all bits without echoing, use IO.RAL!TF.RNE.

IO.RAL is equivalent to IO.RLB ORed with the subfunction bit TF.RAL. The IO.RAL function can be terminated only by a full character count (input buffer full).

**12.3.2.7 IO.RNE** — The IO.RNE function reads terminal input characters without echoing the characters back to the terminal for immediate display. This feature can be used when typing sensitive information (for example, a password or combination).

(Note that the no-echo mode can also be selected with the SF.SMC function; see Table 12-5, bit TC.NEC.)

The IO.RNE function is equivalent to IO.RLB ORed with the subfunction bit TF.RNE.

**12.3.2.8 IO.RPR** — The IO.RPR Read After Prompt functions as an IO.WLB (to write a prompt to the terminal) followed by IO.RLB. However, IO.RPR differs from this combination of functions as follows:

- System overhead is lower with the IO.RPR because only one QIO is processed.
- When using the IO.RPR function, there is no “window” during which a response to the prompt may be ignored. Such a window occurs if IO.WAL/IO.RLB is used, because no read may be posted at the time the response is received.
- If the issuing task is checkpointable, it can be checkpointed during both the prompt and the read requested by the IO.RPR.
- A CTRL/O that may be in effect prior to issuing the IO.RPR is canceled before the prompt is written.



Subfunction bits may be ORed with IO.RPR to write the prompt as a Write All (TF.BIN). In addition, read subfunction bits TF.RAL, TF.RNE, and TF.RST can be used with IO.RPR.

**12.3.2.9 IO.RPR!TF.BIN** — This function results in a read after a “binary” prompt; that is, a prompt is written by the driver with no character interpretation (as if it were issued as an IO.WAL).

**12.3.2.10 IO.RST** — This function is similar to an IO.RLB, except certain special characters terminate the read. These characters are in the ranges 0-037 and 175-177. The driver does not interpret the terminating character, with certain exceptions.<sup>3</sup> For example, a horizontal TAB (011) is not expanded, a RUBOUT (or DEL, 177) does not erase.

Upon successful completion of an IO.RST request that was not terminated by filling the input buffer, the first word of the I/O status block contains the terminating character in the high byte and the IS.SUC status code in the low byte. The second word contains the number of bytes contained in a buffer. The terminating character is not put in the buffer.

IO.RST is equivalent to IO.RLB!TF.RST.

**12.3.2.11 SF.SMC** — This function enables a task to set and reset the characteristics of a terminal. Set Multiple Characteristics is the inverse function of SF.GMC. Like SF.GMC, it is called in the following way:

```
QIO$C SF.SMC,....,<stadd,size>
```

**stadd** The starting address of a buffer of length “size” bytes. Each word in the buffer has the form

```
.BYTE characteristic-name
.BYTE value
```

**characteristic-name** One of the symbolic bit names given in Table 12-5.

**value** Either 0 (to clear a given characteristic) or 1 (to set a characteristic).

Table 12-8 notes the restrictions that apply to these characteristics.

If the characteristic-name is TC.TTP (terminal type), value can have any of the values listed in Table 12-6.

Specifying any value for TC.TBF flushes (clears) the type-ahead buffer (forces the type-ahead buffer count to 0).

3. If upper- to lowercase conversion is disabled, characters 175 and 176 do not act as terminators. CTRL/O, CTRL/Q, and CTRL/S (017, 021, and 023, respectively) are not special terminators. The driver interprets them as output control characters in a normal manner.

**12.3.2.12 IO.RTT** — This QIO function reads characters in a manner like the IO.RLB function, except a user-specified character terminates the read operation. The specified character's code can range from 0–377. It is user designated by setting the appropriate bit in a 16-word table that corresponds to the desired character. Multiple characters can be specified by setting their corresponding bits.

The 16-word table starts at the address specified by the table parameter. The first word contains bits that represent the first 16 ASCII character codes (0–17); similarly, the second word contains bits that represent the next 16 character codes (20–37), and so forth, through the sixteenth word, bit 15, which represents character code 377. For example, to specify the % symbol (code 045) as a read terminator character, set bit 05 in the third word, since the third word of the table contains bits representing character codes 40–57.

If the CTRL/S (023), CTRL/Q (021), and/or any characters whose codes are greater than 177 are desired as the terminator character(s), the terminal must be set to read-pass-all operation (TC.BIN=1), or read-pass 8-bits (TC.8BC), as listed in Table 12-5.

The optional time-out count parameter can be included, as desired.

**12.3.2.13 IO.WAL** — The Write All function causes the driver to pass all output from the buffer without interpretation. It does not intercept control characters. Long lines are not wrapped around if input/output wrap-around has been selected.

IO.WAL is equivalent to the IO.WLB!TF.WAL function.

**12.3.2.14 IO.WSD** — The Write Special Data function is used to communicate nontext information to the terminal task. The buffer address and length are the same as for IO.WLB. The data type parameter indicates to the terminal task what type of data is contained in the buffer. The available data types are:

SD.GDS GIDIS output

Note: This QIO implements a private data path to the terminal subsystem. The interface is subject to change and is, therefore, for DIGITAL use only. The PRO/Graphics and remote terminal emulator should be the only software using IO.WSD.

**12.3.2.15 IO.RSD** — The Read Special Data function is also used in communicating nontext information to the terminal task. The buffer address, length, and timeout are the same as for IO.RLB. The data type parameter indicates to the terminal what type of data is to be read.

Note: This QIO implements a private data path to the terminal subsystem. The PRO/Graphics and remote terminal emulator should be the only software using IO.RSD.

The following restrictions apply to the use of IO.RSD:

- In some ways, IO.RSD is the same as a normal read. One result of this is that if there is a read currently outstanding to the keyboard, the IO.RSD does not take effect until the read to the keyboard is complete.
- While an IO.RSD is pending, no input processing takes place until it completes. So any characters that come in from the keyboard go directly to the typeahead buffer, no ASTs take place, and no characters are echoed.
- When special data comes into the terminal driver from the terminal task (for example, a GIDIS report) and no IO.RSD is outstanding, the special data goes into a special typeahead buffer. That typeahead buffer is capable of holding a maximum of 36 bytes. If more characters are input than the buffer can hold, those characters are discarded and no error message is returned.

If there is an IO.RSD pending when special data comes into the terminal driver from the terminal task, the data goes directly into a read buffer. However, the length of one report may not exceed 36 bytes.

As a result of these restrictions, the recommended way to get a special data report is to first issue the IO.WSD to cause the report to occur and then to issue an IO.RSD for the exact length of the request. This causes IO.RSD to complete immediately, preventing it from blocking the keyboard input.

## 12.4 STATUS RETURNS

Table 12-9 lists error and status conditions that are returned by the terminal driver to the I/O status block.

Most error and status codes returned are byte values. For example, the value for IS.SUC is 1. However, IS.CC, IS.CR, IS.ESC, and IS.ESQ are word values. When any of these codes are returned, the low byte indicates successful completion, and the high byte shows what type of completion occurred.

To test for one of these word-value return codes, first test the low byte of the first word of the I/O status block for the value IS.SUC. Then, test the full word for IS.CC, IS.CR, IS.ESC, IS.ESQ, or IS.CSQ. (If the full word tests equal to IS.SUC, then its high byte is 0, indicating byte-count termination of the read.)

The "error" return IE.EOF may be considered a successful read since characters returned to the task's buffer can be terminated by a CTRL/Z character.

The SE.xxx codes are returned by the SF.GMC and SF.SMC functions as described in Sections 12.3.2.4 and 12.3.2.11. When any of these codes are returned, the low byte in the first word in the I/O status block contains IE.ABO. The second IOSB word contains an offset (starting from 0) to the byte in error in the QIO's stadd buffer.

Table 12-9  
Terminal Status Returns

<i>Code</i>	<i>Reason</i>
IE.EOF	<i>Successful completion on a read with end-of-file</i> The line of input read from the terminal was terminated with the end-of-file character CTRL/Z. The second word of the I/O status block contains the number of bytes read before CTRL/Z was seen. The input buffer contains those bytes.
IS.SUC	<i>Successful completion</i> The operation specified in the QIO directive was completed successfully. If the operation involved reading or writing, you can examine the second word of the I/O status block to determine the number of bytes processed. The input buffer contains those bytes.
IS.CC	<i>Successful completion on a read</i> The line of input read from the terminal was terminated by an INTERRUPT/DO sequence. The input buffer contains the bytes read.
IS.CR	<i>Successful completion on a read</i> The line of input read from the terminal was terminated by a RETURN. The input buffer contains the bytes read.
IS.ESC	<i>Successful completion on a read</i> The line of input read from the terminal was terminated by an Escape character. The input buffer contains the bytes read.
IS.ESQ	<i>Successful completion on a read</i> The line of input read from the terminal was terminated by an escape sequence. The input buffer contains the bytes read and the escape sequence.
IS.PND	<i>I/O request pending</i> The operation specified in the QIO directive has not yet been executed. The I/O status block is filled with zeroes.
IS.TMO	<i>Successful completion on a read</i> The line of input read from the terminal was terminated by a time-out (TF.TMO was set and the specified time interval was exceeded). The input buffer contains the bytes read.
IE.ABO	<i>Operation aborted</i> The specified I/O operation was cancelled by IO.KIL while in progress or while in the I/O queue. The second word of the I/O status block indicates the number of bytes that were put in the buffer before the kill was effected.
IE.BAD	<i>Bad parameter</i> The size of the buffer exceeds 8128 bytes.
IE.DAA	<i>Device already attached</i> The physical device unit specified in an IO.ATT function was already attached by the issuing task. This code indicates that the issuing task has already attached the desired physical device unit, not that the unit was attached by another task. If the attach specified TF.AST or TF.ESQ, these subfunction bits have no effect.
IE.DNA	<i>Device not attached</i> The physical device unit specified in an IO.DET function was not attached by the issuing task. This code has no bearing on the attachment status of other tasks.

Table 12-9 (Cont.)

<i>Code</i>	<i>Reason</i>
IE.DNR	<i>Device not ready</i> The physical device unit specified in the QIO directive was not ready to perform the desired I/O operation. This code is returned to indicate that a time-out occurred on the physical device unit (that is, an interrupt was lost).
IE.IES	<i>Invalid escape sequence</i> An escape sequence was started but escape-sequence syntax was violated before the sequence was completed. (See Section 12.6.4.) The character causing the violation is the last character in the buffer.
IE.IFC	<i>Illegal function</i> A function code specified in an I/O request was illegal for terminals.
IE.NOD	<i>Buffer allocation failure</i> System dynamic storage has been depleted resulting in insufficient space available to allocate an intermediate buffer for an input request or an AST block for an attach request.
IE.PES	<i>Partial escape sequence</i> An escape sequence was started, but read-buffer space was exhausted before the sequence was completed. See Section 12.6.4.3.
IE.SPC	<i>Illegal address space</i> The buffer specified for a read or write request was partially or totally outside the address space of the issuing task, a byte count of 0 was specified, or an odd or 0 AST address was specified.
SE.NIH	A terminal characteristic other than those in Table 12-5 was named in an SF.GMC or SF.SMC request, or a task attempted to assert TC.PRI.
SE.FIX	An attempt was made to change a fixed characteristic in a SF.SMC subfunction request (for example, an attempt was made to change the unit number).
SE.VAL	The new value specified in an SF.SMC request for the TC.TTP terminal characteristic was not one of those listed in Table 12-6.

## 12.5 CONTROL CHARACTERS AND SPECIAL KEYS

This section describes the meanings of the system's special terminal control characters and keys. Note that the driver does not recognize control characters and special keys during a Read All request (IO.RAL), and recognizes only some of them during a Read with Special Terminators (IO.RST).

### 12.5.1 Control Characters

A control character is input from a terminal by holding the control key (CTRL) down while typing one other key. Three of the control characters described in Table 12-10, CTRL/R, CTRL/U, and CTRL/Z, are echoed on the terminal as ^R, ^U, and ^Z, respectively.

Note: The use of control characters on PC 300 Series machines is not recommended except when the PC 300s are connected to another system as a terminal. In normal circumstances use the function keys on the PC300s.

Table 12-10  
Terminal Control Characters

<i>Character</i>	<i>Meaning</i>
CTRL/O	<p>CTRL/O suppresses terminal output. For attached terminals, CTRL/O remains in effect (output is suppressed) until one of the following occurs:</p> <ul style="list-style-type: none"> <li>The terminal is detached.</li> <li>Another CTRL/O character is typed.</li> <li>An IO.CCO or IO.WBT function is issued.</li> <li>Input is entered.</li> </ul> <p>For unattached terminals, CTRL/O suppresses output for only the current output buffer (typically one line).</p>
CTRL/Q	CTRL/Q resumes terminal output previously suspended by means of CTRL/S.
CTRL/S	CTRL/S suspends terminal output. (Output can be resumed by typing CTRL/Q.)
CTRL/R	<p>Typing CTRL/R results in a RETURN and line feed being echoed, followed by the incomplete (unprocessed) input line. Any tabs that were input are expanded and the effect of any rubouts is shown. On hardcopy terminals, CTRL/R allows verifying the effect of tabs and/or rubouts in an input line. CTRL/R is also useful for CRT terminals for the CRT rubout. For example, after rubbing out the left-most character on the second displayed line of a wrapped input line, the cursor does not move to the right of the first displayed line. In this case, CTRL/R brings the input line and the cursor back together again.</p>
CTRL/U	Typing CTRL/U before typing a line terminator deletes previously typed characters back to the beginning of the line. The system echoes this character as ^U followed by a RETURN and a line feed.
CTRL/X	This character clears the type-ahead buffer.
CTRL/Z	CTRL/Z indicates an end-of-file for the current terminal input.
	<p>Note: On the PC300 series systems, the Hold Screen key should be used instead of CTRL/S and CTRL/Q.</p>

### 12.5.2 INTERRUPT/DO AST Information

If the application has done an IO.ATA QIO without specifying the TF.XCC subfunction, the following will happen:

<i>Key One</i>	<i>Key Two</i>	<i>Result</i>
non Interrupt key	Do key	The non-Interrupt key is handled as usual and the application gets the escape sequence for the Do key
Interrupt key	non Do key	The Interrupt key is discarded and the non Do key is handled as if Interrupt had not been pressed
Interrupt key	Do key	Applications AST routine is activated just as if a CTRL/C was typed
CTRL/C		Applications AST routine is activated as for RSX-11M-PLUS

If the application has not done an IO.ATA QIO or if it has done one with the TF.XCC subfunction bit set, the following will happen:

<i>Key One</i>	<i>Key Two</i>	<i>Result</i>
non Interrupt key	Do key	The non Interrupt key is handled as usual and the application gets the escape sequence for the Do key
Interrupt key	non Do key	The Interrupt key is discarded and the non Do key is handled as if the Interrupt key had not been pressed
Interrupt key	Do key	The PRO/Dispatcher is notified; aborts all application tasks. The application gets no indication that anything happened
CTRL/C		The PRO/Dispatcher is notified; the application gets no indication that anything happened

If an application puts the terminal in Read Pass All mode or if it specifies TF.RAL on a read, all keys, except for Hold Screen and Print Screen, will go into the type-ahead buffer unprocessed. The Interrupt and Do keys will go in as the escape sequences that they represent.

Any characters for which there is no read or AST request outstanding will be put into the type-ahead buffer. The buffer is 36 bytes long. If the buffer is full, and if a character is typed that would go into the type-ahead buffer, a bell is echoed and the character is discarded. When either CTRL/C or the INTERRUPT/DO sequence is entered, the type-ahead buffer is flushed.

### 12.5.3 Special Keys

The RETURN, and DELETE keys have special significance for terminal input, as described in Table 12-11. A line can be terminated by a RETURN, or CTRL/Z characters, or by completely filling the input buffer—that is, by exhausting the byte count before a line terminator is typed. The standard buffer size for a terminal can be determined for a task by issuing a Get LUN Information system directive and examining Word 5 of the buffer.

## 12.6 ESCAPE SEQUENCES

Escape sequences are strings of two or more characters beginning with an ESC (033) character.

Escape sequences provide a way to pass input to a task without interpretation by the operating system. This could be done with a number 1-character Read All functions, but escape sequences allow them to be read with IO.RLB requests.

Table 12-11  
Special Terminal Keys

<i>Key</i>	<i>Meaning</i>
RETURN	Typing RETURN terminates the current line and causes the carriage or cursor to return to the first column on the line.
DELETE	Typing DELETE deletes the last character typed on an input line. Only characters typed since the last line terminator may be deleted. Several characters can be deleted in sequence by typing successive DELETES.  DELETE causes the last typed character (if any) to be removed from the incomplete input line and a backspace-space-backspace sequence of characters for that terminal are echoed. If the last typed character was a tab, enough backspaces are issued to move the cursor to the character position before the tab was typed. If a long input line was split, or "wrapped," by the automatic-return option, and a DELETE erases the last character of a previous line, the cursor is not moved to the previous line.



### 12.6.1 Definition

The format of an escape sequence as defined in American National Standard X 3.41 – 1974 and used in the VT100 is:

ESC ... F

- ESC The introducer control character ( $33_8$ ) that is named escape.
- ... The intermediate bit combinations that may or may not be present. I characters are bit combination  $40_8$  to  $57_8$  inclusive in both 7- and 8-bit environments.
- F The final character. F characters are bit combinations  $60_8$  to  $176_8$  inclusive in escape sequences in both 7- and 8-bit environments.

The occurrence of characters in the inclusive ranges  $0_8$  to  $37_8$  is technically an error condition whose recovery is to execute immediately the function specified by the character and then continue with the escape sequence execution. The exceptions are: if the character ESC occurs, the current escape sequence is aborted, and a new one commences, beginning with the ESC just received; if the character CAN ( $30_8$ ) or the character SUB ( $32_8$ ) occurs, the current escape sequence is aborted, as is the case with any control character.

### 12.6.2 Prerequisites

Two prerequisites must be satisfied before escape sequences can be received by a task.

1. The task must “ask” for them by issuing an IO.ATT function and invoking the subfunction bit TF.ESQ.
2. The terminal must be declared capable of generating escape sequences. A way to tell the driver that the terminal can generate escape sequences is by issuing the Set Multiple Characteristics request. (See Section 12.3.2.11).

If these prerequisites are not satisfied, the ESC character is treated as a line terminator.

### 12.6.3 Characteristics

Escape sequences always act as line terminators. That is, an input buffer may contain other characters that are not part of an escape sequence, but an escape sequence always comprises the last characters in the buffer.

Escape sequences are not echoed. However, if a non-CRT DELETE sequence is in progress, it is closed with a backslash when an escape sequence is begun.

Escape sequences are not recognized in unsolicited input streams. Neither are they recognized in a Read with Special Terminators (subfunction bit TF.RST) nor in a Read All (subfunction bit TF.RAL).

### 12.6.4 Escape Sequence Syntax Violations

A violation of the syntax defined in Section 12.6.1 causes the driver to abandon the escape sequence and to return an error (IE.IES).

**12.6.4.1 DEL (177)** — The character DELETE is not legal within an escape sequence. If it occurs at any point within an escape sequence, the entire sequence is abandoned and deleted from the input buffer.

**12.6.4.2 Control Characters (0-037)** — The reception of any character in the range 0 to 037 (with four exceptions) is a syntax violation that terminates the read with an error (IE.IES). Four control characters are allowed: CTRL/Q, CTRL/S, CTRL/X, and CTRL/O. These characters are handled normally by the operating system even when an escape sequence is in progress.

**12.6.4.3 Full Buffer** — A syntax error results when an escape sequence is terminated by running out of read-buffer space, rather than by receipt of a final character. The error IE.PES is returned. For example, after a task issues an IO.RLB with a buffer length of 2, and the following characters are entered:

```
ESC!A
```

the buffer contains "ESC!", and the I/O status block contains:

```
IOSB      IE.PES
          2
```

The "A" is treated as unsolicited input.

## 12.7 VERTICAL FORMAT CONTROL

Table 12-12 is a summary of all characters used for vertical format control on the terminal. Any one of these characters can be specified as the value of the vfc parameter in IO.WLB, IO.WVB, IO.WBI, IO.CCO, or IO.RPR functions.

Table 12-12  
Vertical Format Control Characters

<i>Octal Value</i>	<i>Character</i>	<i>Meaning</i>
040	blank	SINGLE SPACE—Output one line feed, print the contents of the buffer, and output a RETURN. Normally, printing immediately follows the previously printed line.
060	0	DOUBLE SPACE—Output two line feeds, print the contents of the buffer, and output a RETURN. Normally, the buffer contents are printed two lines below the previously printed line.
061	1	PAGE EJECT—If the terminal supports FORM FEEDs, output a form feed, print the contents of the buffer, and output a RETURN. If the terminal does not support FORM FEEDs, the driver simulates the FORM FEED character by either outputting four line feeds to a crt terminal, or by outputting enough line feeds to advance the paper to the top of the next page on a printing terminal.
053	+	OVERPRINT—Print the contents of the buffer and output a RETURN, normally overprinting the previous line.
044	\$	PROMPTING OUTPUT—Output one line feed and print the contents of the buffer. This mode of output is intended for use with a terminal on which a prompting message is output, and input is then read on the same line.
000	null	INTERNAL VERTICAL FORMAT—Print the buffer contents without addition of vertical format control characters.

All other vertical format control characters are interpreted as blanks (040).

A task can determine the buffer width by issuing a Get LUN Information directive and examining word 5 returned in the buffer.

It is possible to lose track of where you are in the input buffer if wrap-around is enabled for your terminal. For example, while deleting text on a wrapped line, the cursor will not back up to the previous line.

## 12.8 TYPE-AHEAD BUFFERING

Characters received by the terminal driver are either processed immediately or stored in the type-ahead buffer. The type-ahead buffer allows characters to be temporarily stored and retrieved FIFO. The type-ahead buffer is used as follows:

### 1. Store in buffer:

An input character is stored in the type-ahead buffer if one or more of the following conditions are true:

- There is at least one character presently in the type-ahead buffer.
- The character input requires echo and the output line to the terminal is presently busy outputting a character.
- No read request is in progress, no unsolicited input AST is specified. A character is not echoed when it is stored in the buffer. Echoing a character is deferred until it is retrieved from the buffer, since the read mode (for example, read-without-echo) is not known by the driver until then.

Note: Depending on the terminal mode and the presence of a read function, read subfunctions and an unsolicited input AST, the INTERRUPT/DO, CTRL/O, CTRL/Q, CTRL/S, and CTRL/X characters may be processed immediately and not stored in the type-ahead buffer.

### 2. Retrieve from buffer:

When the driver becomes ready to process input, or when a task issues a read request, an attempt is made to retrieve a character from the buffer. If this attempt is successful, the character is processed and echoed, if required. The driver then loops, retrieving and processing characters until either the buffer is empty, the driver becomes unable to process another character, or a read request is finished with the terminal attached or slaved.

### 3. Flush the buffer:

The buffer is flushed (cleared) when:

- CTRL/X is received.
- INTERRUPT/DO is received.

Exceptions: CTRL/X does not flush the buffer if read-pass-all or read-with-special-terminators is in effect.

If the buffer becomes full, each character that cannot be entered causes a BELL character to be echoed to the terminal.

If a character is input and echo is required, but the transmitter section is busy with an output request, the input character is held in the type-ahead buffer until output (transmitter) completion occurs.

## 12.9 FULL-DUPLEX OPERATION

When a terminal line is in the full-duplex mode, the full-duplex driver attempts to simultaneously service one read request and one write request. The Attach, Detach and Set Multiple Characteristics functions are only performed with the line in an idle state (not executing a read or a write request).

## 12.10 INTERMEDIATE INPUT AND OUTPUT BUFFERING

Input buffering for checkpointable tasks with checkpointing enabled is provided in the terminal driver private pool. As each buffer becomes full, a new buffer is automatically allocated and linked to the previous buffer. The Executive then transfers characters from these buffers to the task buffer and the terminal driver deallocates the buffers once the transfer has been completed.

If the driver fails to allocate the first input buffer, the characters are transferred directly into the task buffer. If the first buffer is successfully allocated, but a subsequent buffer allocation fails, the input request terminates with the error code IE.NOD. In this case, the I/O status block contains the number of characters actually transferred to the task buffer. The task may then update the buffer pointer and byte count and reissue a read request to receive the rest of the data. The type-ahead buffer ensures that no input data is lost as long as the type-ahead buffer is not full.

All terminal output is buffered. As many buffers as required are allocated by the terminal driver and linked to a list. If not enough buffers can be obtained for all output data, the transfer is done as a number of partial transfers, using available buffers for each partial transfer. This is transparent to the requesting task. If no buffers can be allocated, the request terminates with the error code IE.NOD.

The unconditional output buffering serves two purposes:

1. It reduces time spent at system state.
2. It enables task checkpointing during the transfer to the terminal (if all output fits in one buffer list).

## 12.11 TERMINAL-INDEPENDENT CURSOR CONTROL

The terminal driver responds to task I/O requests for cursor positioning without the task requiring information about the type of terminal in use. I/O functions associated with cursor positioning are described as follows.

Cursor position is specified in the vfc parameter of the IO.WLB or IO.RPR function. The parameter is interpreted simply as a vfc parameter if the high byte of the parameter is 0. However, if the parameter is used to define cursor position, the high byte must be nonzero, the low byte is interpreted as column number (x-coordinate), and the high byte is interpreted as line number (y-coordinate).

Home position, the upper left corner of the display, is defined as 1,1. Depending upon terminal type, the driver outputs appropriate cursor-positioning commands appropriate for the terminal in use that will move the cursor to the specified position. If the most significant bit of the line number is set, the driver clears the display before positioning the cursor.

When defining cursor position in an IO.WLB function, the TF.RCU subfunction can be used to save the current cursor position. When included in this manner, TF.RCU causes the driver to first save the current cursor position, then position the cursor and output the specified buffer, and, finally, restore the cursor to the original (saved) position once the output transfer has been completed.

## 12.12 PROGRAMMING HINTS

Using IO.WVB instead of IO.WLB is recommended when writing to a terminal. If the write actually goes to a terminal, the Executive converts the IO.WVB to an IO.WLB request. However, if the LUN has been redirected to an appropriate device—a disk, for example—the use of an IO.WVB function will be rejected because a file is not open on the LUN. This prevents privileged tasks from overwriting block zero of the disk.

Note that any subfunction bits specified in the IO.WVB request (for example, TF.CCO, TF.WAL, TF.WBT) are stripped when the IO.WVB is converted to IO.WLB.

# CHAPTER 13

## THE KERNEL COMMUNICATIONS DRIVER

---

### 13.1 INTRODUCTION

The Professional 300 kernel communications driver (XK) permits use of the kernel communication port in asynchronous mode. The XK driver provides the following features:

- Full duplex operation
- Input buffering
- Unsolicited event AST's
- Transfer length of up to 8128 bytes
- Optional time-out on solicited input
- Optional XON/XOFF support
- Modem support

**13.2 GET LUN INFORMATION MACRO**

Word 2 of the buffer filled by the Get LUN Information system directive (the first characteristics word) contains the information noted in Table 13-1 for the XK driver. A setting of 1 indicates that the described characteristic is true.

Table 13-1

<i>Bit</i>	<i>Setting</i>	<i>Meaning</i>
0	0	Record-oriented device
1	0	Carriage-control device
2	0	Terminal device
3	0	File structured device
4	0	Single-directory device
5	1	Sequential device
6	0	Mass storage device
7	0	User-mode diagnostics supported, device dependent
8	0	Device supports 22-bit direct addressing
9	0	Unit software write-locked
10	0	Input spooled device
11	0	Output spooled device
12	0	Pseudo device
13	0	Device mountable as communications channel
14	0	Device mountable as a FILES-11 volume
15	0	Device mountable

Words 3 and 4 of the buffer are undefined. Word 5 indicates size of the internal input ring buffer.



### 13.3 QIO MACRO

Table 13-2 lists the standard and device-specific functions of the QIO macro that are valid for the XK driver.

Table 13-2  
Standard and Device-Specific QIO Functions

<i>Format</i>	<i>Function</i>
<i>Standard Functions</i>	
QIO\$ CIO.DET,...	Detach device.
QIO\$C IO.KIL,...	Cancel I/O requests.
QIO\$C IO.RLB,...,<stadd,size[,tmo]>	Read logical block (read input into buffer).
QIO\$C IO.RVB,...,<stadd,size[,tmo]>	Read virtual block (read input into buffer).
QIO\$C IO.WLB,...,<stadd,size>	Write logical block (send contents of buffer).
QIO\$C IO.WVB,...,<stadd,size>	Write virtual block (send contents of buffer).
<i>Device-Specific Functions</i>	
QIO\$C IO.ANS,...,<stadd,size>	Initiate a connection in answer mode, either in response to a ringing line, or if a connection already exists.
QIO\$C IO.ATA,...,<ast[,par2]>	Attach device, specify unsolicited event AST.
QIO\$C IO.BRK,...,<type>	Send a BREAK.
QIO\$C IO.CON,...,<stadd,size[,tmo]>	Dial and connect.
QIO\$C SF.GMC,...,<stadd,size>	Get multiple characteristics.
QIO\$C IO.HNG,...	Hang up a line.
QIO\$C IO.LTI,...,<stadd,size[,par3]>	Connect for unsolicited event AST's while detached.
QIO\$C IO.ORG,...,<stadd,size[,tmo]>	Initiate a connection in originate mode, assuming the line has already been connected.
QIO\$C IO.RAL,...,<stadd,size[,tmo]>	Read logical block, pass all bits.
QIO\$C IO.RNE,...,<stadd,size[,tmo]>	Read logical block, do not echo.
QIO\$C SF.SMC,...,<stadd,size>	Set multiple characteristics.
QIO\$C IO.TRM,...	Unload driver.
QIO\$C IO.UTI,...	Disable unsolicited event AST's while detached.
QIO\$C IO.WAL,...,<stadd,size>	Write logical block, pass all bits.

ast	the entry point for an unsolicited event AST
par2	
par3	a number that can be used to identify this line as the input source upon entry to an unsolicited event AST routine
size	the size of the stadd data buffer in bytes. The specified size must be greater than zero and less than or equal to 8128; the buffer must be within the task's address space
stadd	the starting address of the data buffer; the address may be byte aligned
type	either 0 or 1 to indicate either a break or a long space
tmo	an optional time-out count when used in conjunction with TF.TMO on read requests, IO.CON, and IO.ORG requests

The time-out is specified as follows:

```
.BYTE x,y
```

where x is the number of ten-second intervals, up to 255<sub>10</sub>, and y is the number of one-second interval, also up to 255<sub>10</sub>. The longest possible time-out interval that can be specified is 255<sub>10</sub> seconds. If the time-out value is larger than 255 seconds, 255 seconds is used. Section 13.7 describes the effect of the time-out parameters on specific requests.

### 13.3.1 Device-Specific QIO Functions

Several of the device-specific functions described in this section can be issued by ORing a particular subfunction bit with another QIO function. These subfunction bits are specified in the following descriptions.

**13.3.1.1 IO.ANS** —The Answer function establishes a connection in answer mode, either in response to a ringing line, or if connection already exists. If a connection is not complete within 30 seconds, an IE.DNR error will be returned. The buffer address is required but is not used.

**13.3.1.2 O.ATA** —IO.ATA is a variation of the Attach function. IO.ATA specifies an asynchronous system trap (AST) to process unsolicited events when called as follows:

```
QIO$C IO.ATA,...,<ast[,par2]>
```

When an unsolicited event occurs, the resulting AST serves as notification of the unsolicited event. Upon entry to the AST, the high byte of the top word on the stack contains par2, if it was specified. The low byte contains the event type. This word must be removed from the stack before exiting the AST. See section 13.6 for more information on unsolicited events.

*IO.ATA is equivalent to IO.ATT ORed with the subfunction bit TF.AST.*

**13.3.1.3 IO.BRK** —When issued, the IO.BRK function causes either a break or a long space to be sent. If parameter 1 is zero, a break is sent. If parameter 1 is one, a long space is sent.

On the kernel communication port, a break will last for approximately 235 milliseconds, and a long space approximately 3.5 seconds.

**13.3.1.4 I.O.CON** —The IO.CON function dials and connects a line in originate mode, as follows:

```
QIO$C IO.CON,...,<stadd,size[,tmo]>
```

where stadd is the address of the telephone number to dial.

If TF.TMO is not specified, the request will complete when a connection is established or after 60 seconds.

**13.3.1.5 SF.GMC** —The Get Multiple Characteristics function returns driver characteristics information, as follows:

```
QIO$C SF.GMC,...,<stadd,size>
```

where stadd is the starting address of a data buffer of length "size" bytes. Each word in the buffer has the form:

```
.BYTE characteristic-name
.BYTE 0
```

where characteristic-name is one the bit names given in Table 13-3 . The value returned in the high byte of each byte-pair is value of that characteristic.

Table 13-3  
XK Driver Characteristics for SF.GMC and SF.SMC Functions

<i>Bit Name</i>	<i>Valid Values</i>	<i>Meaning</i>
TC.ARC	0-9.	Auto-answer ring count (0 => don't answer)
TC.BIN	0,1	Enable or disable XON/XOFF support
TC.CTS	0,1	Resume or suspend output
TC.EPA	0,1	Odd or Even parity (if TC.PAR is specified)
TC.FSZ	Note 1	Character width including parity (if any)
TC.PAR	0,1 Note 1	Enable parity checking and generation
TC.RSP	Note 2	Receiver speed (bits-per-second)
TC.STB	1,2	Number of stop bits
TC.TBF	Note 3	Input ring buffer count or flush
TC.TRN	Note 4	Set translate table
TC.XMM	0,1	Disable or enable Maintenance mode
TC.XSP	Note 2	Transmitter speed (bits-per-second)
TC.8BC	0,1 Note 1	Pass 8-bit characters on input and output
XT.MTP	Note 5	Modem type

1. TC.FSZ is the frame size of a character. It is the number of data bits per character, plus 1 if parity is enabled.

TC.FSZ and TC.PAR interact with each other to determine the number of data bits returned to the task. Table 13-4 shows the relationship of these characteristics.

Two combinations do not appear in Table 13-4; TC.FSZ=9 with TC.PAR=0, and TC.FSZ=5 with TC.PAR=1. These two combinations are invalid, and the driver will return an error. To avoid this problem, always set the value of TC.FSZ first. The driver will automatically enable or disable parity if the value of TC.FSZ is 9 or 5.

If the value of TC.FSZ is 8 or 9, the number of data bits returned to the task is further modified by the value of TC.8BC. If TC.8BC is set to 1, all 8 data bits will be returned to the task. If TC.8BC is set to 0, only 7 data bits will be returned to the task. Setting TC.BIN to a value of 1 or using the IO.RAL function will override the value of TC.8BC.

## 2. TC.RSP and TC.XSP values and corresponding baud rates are:

Table 13-5  
Receiver and Transmitter Speed Values (TC.RSP, TC.XSP)

<i>TC.RSP or TC.XSP Value</i>	<i>Actual Baud Rate (in bits-per-second)</i>
S.50	50
S.75	75
S.110	110
S.134	134.5
S.150	150
S.200	200
S.300	300
S.600	600
S.1200	1200
S.1800	1800
S.2000	2000
S.2400	2400
S.3600	3600
S.4800	4800
S.7200	7200
S.9600	9600
S.19.2	19200

3. The TC.TBF characteristic returns the number of unprocessed characters in the input buffer when used with SF.GMC. If there are more than 255 characters in the buffer, the value 255 will be returned. When used with SF.SMC, TC.TBF causes the input buffer to be flushed.
4. The translate table allows translation for either non-standard pulsing arrangements or for modems other than the DF03 which may be connected to the XK Communications port. The translate table is made up of three sections; a dial translation table, a start sequence string, and an end sequence string. Any or all sections of the translate table may be empty.

The format of this characteristic is:

```
.BYTE          TC.TRN,count1,count2,count3
.BYTE chr1,rep_chr1      ;Translate table portion.
.              ;1st char is the char to
.              ;translate, 2nd char is
.              ;the replacement char
.BYTE chrn,rep_chrn      End of translate table
.BYTE ss1,ss2, . . .     ;ss1, ss2, . . . are the
                        ;start characters
.BYTE es1,es2, . . .     ;es1, es2, . . . are the
                        ;end characters
.EVEN
```

where count1 is the length of the dial translate table, count2 is the length of the start sequence, and count3 is the length of the end sequence.

count2 and/or count3 can be zero if you do not specify a start and/or end sequence.

The QIO buffer count in the IO.SMC includes the bytes containing the TC.TRN characteristic and the count bytes.

The dial translate table is a string of character pairs, input character followed by output character. This translate table is used to convert a telephone number, typically to remove format effectors such as "(", ")", "-", and " ". If a character in the telephone number matches a character in the input section of the dial translate table, the character is converted to the character from the output section. If the character from the output section is 0, the character from the telephone number will be ignored.

The start sequence string, if specified, will be sent to the autodialer before the phone number.

The end sequence string, if specified, will be sent to the autodialer after the phone number.

Note:

1. The next characteristic must begin on a word boundary.
2. This is a write-only parameter, and will return an SE.NIH error if used with the SF.GMC function.

5. The XT.MTP characteristic has the following values:

```
XTM.NO          No modem, hard-wired line
XTM.FS          USFSK - 0..300 baud Bell 103J
XTM.21          CCITTV.21 - 0..300 baud European
XTM.M1          CCITTV.23 Mode 1 - 75/0..300 split baud
XTM.M2          CCITTV.23 Mode 2 - 75/0..1200 split baud
XTM.PS          DPSK - 1200 baud Bell 212
XTM.US          Mini Exchange
```

**13.3.1.6 IO.HNG**—The IO.HNG function causes a line to be hung up.

**13.3.1.7 IO.LTI**—The IO.LTI function causes the driver to deliver unsolicited event notification AST's to a specified task, if the driver is not attached by another task. It is called as follows:

```
QIO$C IO.LTI,....,<stadd,size[,par3]>
```

where stadd is the address of a three word buffer of the form:

```
.WORD  AST_address
.RAD50  /first_half_of_task_name/
.RAD50  /second_half_of_task_name/
```

When an unsolicited event occurs, the resulting AST serves as notification of the unsolicited event. Upon entry to the AST, the high byte of the top word on the stack contains par3, if it was specified. The low byte contains the event type. This word must be removed from the stack before exiting the AST. See section 13.6 for more information on unsolicited events.





**13.3.1.7 IO.LTI** —The IO.LTI function causes the driver to deliver unsolicited event notification AST's to a specified task, if the driver is not attached by another task. It is called as follows:

```
QIO$C IO.LTI,...,<stadd,size[,par3]>
```

where stadd is the address of a three word buffer of the form:

```
.WORD AST_address
.RAD50 /first_half_of_task_name/
.RAD50 /second_half_of_task_name/
```

When an unsolicited event occurs, the resulting AST serves as notification of the unsolicited event. Upon entry to the AST, the high byte of the top word on the stack contains par3, if it was specified. The low byte contains the event type. This word must be removed from the stack before exiting the AST. See section 13.6 for more information on unsolicited events.

**13.3.1.8 IO.ORG** —The IO.ORG function initiates a connection in originate mode, assuming the line has already been connected. The buffer address is required but is not used.

**13.3.1.9 IO.RAL** —The Read All function causes the driver to pass all bits to the requesting task, when the value of TC.FSZ is 8 or 9. The driver does not mask out the high-order bit. This function is used to temporarily bypass the setting of the TC.8BC characteristic. Note that unlike the RSX-11M/M-PLUS terminal driver, this function does not pass CTRL/Q or CTRL/S to the requesting task. The TC.BIN must be set for these characters to be returned to the task.

IO.RAL is equivalent to IO.RLB ORed with the subfunction bit TF.RAL.

**13.3.1.10 IO.RNE** —The Read with No Echo function is accepted by the driver and the subfunction bit is ignored.

IO.RNE is equivalent to IO.RLB ORed with the subfunction bit TF.RNE.

**13.3.1.11 SF.SMC** —This function enables a task to set and reset the characteristics of the XK driver. Set Multiple Characteristics is the inverse function of SF.GMC. Like SF.GMC, is called in the following way:

```
QIO$C SF.SMC,...,<stadd,size>
```

where stadd is the starting address of a data buffer of length "size" bytes. Each word in the buffer has the form:

```
.BYTE characteristic-name
.BYTE value
```

where characteristic-name is one the bit names given in Table 13-2, and value is a value in the range given in Table 13-2.

**13.3.1.12 IO.TRM** —The IO.TRM function causes the driver to be unloaded. The task must be attached to issue this function.

**13.3.1.13 IO.UTI** —The IO.UTI function disables unsolicited event notification while the driver is not attached.

**13.3.1.14 IO.WAL** —The Write All function is accepted by the driver and the subfunction bit is ignored. The driver transmits all data bits in all cases.

IO.WAL is equivalent to IO.WLB ORed with the subfunction bit TF.WAL.

## 13.4 STATUS RETURNS

Table 13-6 lists error and status conditions that are returned by the communications driver to the I/O status block.

The SE.xxx codes are returned by the SF.GMC and SF.SMC functions as described in Sections 13.3.1.5 and 13.3.1.11. When any of these codes are returned, the low byte in the first word of the I/O status block will contain IE.ABO. The second IOSB word contains an offset (starting from 0) to the byte in error in the QIO's stadd buffer.

Table 13-6  
XK Driver Status Returns

<i>Code</i>	<i>Reason</i>
IS.SUC	<i>Successful completion</i> The operation specified in the QIO directive was completed successfully. If the operation involved reading or writing, you can examine the second word of the I/O status block to determine the number of bytes processed. The input buffer contains those bytes.
IS.PND	<i>I/O request pending</i> The operations specified in the QIO directive has not yet been executed. The I/O status block is filled with zeros.
IS.TMO	<i>Successful completion on a read</i> The input from the communications port was terminated by a time-out (non-zero value specified for the tmo parameter). The input buffer contains the bytes read.
IE.ABO	<i>Operation aborted</i> The specified I/O operation was cancelled by IO.KIL while in progress or while in the I/O queue. The second word of the I/O status block indicates the number of bytes that were put in the buffer before the kill was effected.
IE.ALC	<i>Allocation failure</i> The total size of the phone number specified by an IO.CON request plus the start and end sequences was larger than the driver's internal buffer.
IE.CNR	<i>Connection rejected</i> Carrier was already present when an IO.CON request was issued.

Table 13-6 (Cont.)

<i>Code</i>	<i>Reason</i>
IE.DAA	<i>Device already attached</i> The physical device-unit specified in the IO.ATT function was already attached by the issuing task. This code indicates that the issuing task has already attached the desired physical device-unit, not that the unit was attached by another task. If the attach specified TF.AST, the subfunction bit has no effect.
IE.DNA	<i>Device not attached</i> The physical device-unit specified in an IO.DET or IO.TRM function was not attached by the issuing task. This code has no bearing on the attachment status of other tasks.
IE.DNR	<i>Device not ready</i> The physical device-unit specified in the QIO directive was not ready to perform the desired I/O operation. This code is returned to indicate that an attempt was made to perform a function on a line connected to a modem without carrier present, or to indicate that a connection was not established within the time-out period specified by an IO.CON, IO.ANS, or IO.ORG request.
IE.IFC	<i>Illegal function</i> A function code specified in an I/O request was illegal for the communications port.
IE.OFL	<i>Device off-line</i> The physical device-unit associated with the LUN specified in the QIO directive was not online.
SE.NIH	<i>Characteristic not implemented</i> A characteristic other than those specified in Table 13-3 was named in an SF.GMC or SF.SMC request.
SE.VAL	<i>Illegal characteristic value</i> The new value specified in an SF.SMC request was not one of those listed in Table 13-3.

### 13.5 FULL-DUPLEX OPERATION

The XK driver attempts to simultaneously service one read request and one write request. Note that unlike the RSX-11M/M-PLUS full-duplex terminal driver, the SF.SMC function is NOT blocked until the line is idle. Resetting characteristics during I/O operations may cause unpredictable results.

## 13.6 UNSOLICITED EVENT PROCESSING

If a task attaches for unsolicited event AST's (IO.ATA), an AST will be dispatched whenever any of the events listed in Table 13-5 occur. When the AST is entered, the event type will be in the low byte of the top word of the stack, and par2 (IO.ATA) or par3 (IO.LTI) will be in the high byte. Note that the XTU.UI event is processed differently from the rest.

Table 13-7  
Unsolicited Event Types

XTU.CD	Carrier detect
XTU.CL	Carrier loss
XTU.OF	XOFF received
XTU.ON	XON received
XTU.RI	Ring
XTU.UI	Unsolicited input

### 13.6.1 XTU.UI

If the event type is XTU.UI (unsolicited input), the AST becomes "disarmed" until a read request is issued by the task. Once the read request has completed, the AST is "armed" again for new unsolicited events.

## 13.7 TIME-OUT

The optional time-out parameter on read, IO.CON, and IO.ORG requests effects the action of the request. The following sections describe those effects.

### 13.7.1 Read Requests

- tmo = 0      The request completes immediately after transferring as many characters as are available, less than or equal to the size parameter. The number of bytes transferred is returned in the second I/O status word.
- tmo <> 0    The request completes after the time-out period or the requested number of bytes has been transferred. The number of bytes transferred will be returned in the second word of the I/O status block.

**13.7.2 IO.CON**

- tmo = 0        The request completes immediately. If carrier is not present after 60 seconds, DTR and RTS are dropped.
- tmo <> 0      The request completes after the time-out period, or after a connection is established. If the time-out period expires, DTR and RTS will be dropped, and an IE.DNR error will be returned in the first word of the I/O status block. If carrier comes up before the time-out period expires IS.SUC will be returned in the first word of the I/O status block.

**13.7.3 IO.ORG**

- tmo = 0        The request completes immediately. If carrier is down, DTR and RTS will be dropped and an IE.DNR error will be returned in the first word of the I/O status block. If carrier is up, IS.SUC will be returned in the first word of the I/O status block.
- tmo <> 0      The request completes after the time-out period, or after a connection is established. If the time-out period expires, DTR and RTS will be dropped, and an IE.DNR error will be returned in the first word of the I/O status block. If carrier is up (or comes up before the time-out period expires), IS.SUC will be returned in the first word of the I/O status block.

**13.8 XON/XOFF SUPPORT**

If XON/XOFF support is requested (TC.BIN = 0), the driver will transmit an XOFF whenever the ring buffer is three-quarters filled, and an XON whenever the buffer is then emptied below the one-quarter point. Because of this, tasks should not pass XON/XOFF control characters to the driver for transmission.

If an XOFF is received, transmission will be blocked. If the task is attached for unsolicited event AST's, an XTU.OF event will be dispatched. In any case, the TC.CTS parameter will reflect the XON/XOFF state of the line.

If XON/XOFF support is not requested (TC.BIN = 1), and the value of XT.MTP is XTM.NO (no modem), the Clear to Send line will be used in place of XON/XOFF control characters. State changes of this line will cause unsolicited event AST's, and modify the value of TC.CTS.



## APPENDIX A

### STANDARD ERROR CODES

---

The symbols listed below are associated with the directive status codes returned by the Executive. They are determined (by default) at task-build time. To include these in a MACRO-11 program, use the following two lines of code:

```
.MCALL DRERR$
DRERR$
;
; STANDARD ERROR CODES RETURNED BY DIRECTIVES IN THE DIRECTIVE STATUS
; WORD
;
IS.CLR +00    EVENT FLAG WAS CLEAR
IS.SUC +01    OPERATION COMPLETE, SUCCESS
IS.SET +02    EVENT FLAG WAS SET
;
;
;
IE.UPN -01.   INSUFFICIENT DYNAMIC STORAGE
IE.INS -02.   SPECIFIED TASK NOT INSTALLED
IE.UNS -04.   INSUFFICIENT DYNAMIC STORAGE FOR SEND
IE.ULN -05.   UNASSIGNED LUN
IE.HWR -06.   DEVICE DRIVER NOT RESIDENT
IE.ACT -07.   TASK NOT ACTIVE
IE.ITS -08.   DIRECTIVE INCONSISTENT WITH TASK STATE
IE.FIX -09.   TASK ALREADY FIXED/UNFIXED
IE.CKP -10.   ISSUING TASK NOT CHECKPOINTABLE
IE.TCH -11.   TASK IS CHECKPOINTABLE
IE.RBS -15.   RECEIVE BUFFER TOO SMALL
IE.PRI -16.   PRIVILEGE VIOLATION
IE.RSU -17.   SPECIFIED VECTOR ALREADY IN USE
IE.NSW -18.   NO SWAP SPACE AVAILABLE
IE.ILV -19.   SPECIFIED VECTOR ILLEGAL
;
;
;
```

A-2 APPENDIX A: STANDARD ERROR CODES

IE.AST	-80.	DIRECTIVE ISSUED/NOT ISSUED FROM AST
IE.MAP	-81.	ISR OR ENABLE/DISABLE INTERRUPT ROUTINE NOT WITHIN 4K WORDS FROM VALUE OF BASE ADDRESS & 177700
IE.IOP	-83.	WINDOW HAS I/O IN PROGRESS
IE.ALG	-84.	ALIGNMENT ERROR
IE.WOV	-85.	ADDRESS WINDOW ALLOCATION OVERFLOW
IE.NVR	-86.	INVALID REGION ID
IE.NVW	-87.	INVALID ADDRESS WINDOW ID
IE.ITP	-88.	INVALID TI PARAMETER
IE.IBS	-89.	INVALID SEND BUFFER SIZE (>255.)
IE.LNL	-90.	LUN LOCKED IN USE
IE.IUI	-91.	INVALID UIC
IE.IDU	-92.	INVALID DEVICE OR UNIT
IE.ITI	-93.	INVALID TIME PARAMETERS
IE.PNS	-94.	PARTITION/REGION NOT IN SYSTEM
IE.IPR	-95.	INVALID PRIORITY (>250.)
IE.ILU	-96.	INVALID LUN
IE.IEF	-97.	INVALID EVENT FLAG NUMBER
IE.ADP	-98.	PART OF DPB OUT OF USER'S SPACE
IE.SDP	-99.	DIC OR DPB SIZE INVALID



## APPENDIX B

# SUMMARY OF I/O FUNCTIONS

---

This appendix summarizes legal I/O functions for all device drivers described in this manual. Both devices and functions are listed alphabetically. The meanings of the function-specific parameters shown below are discussed in the appropriate driver chapters. The user may reference these functions symbolically by invoking the system macros FILIO\$ (standard I/O functions) and SPCIO\$ (special I/O functions), or by allowing them to be defined at task-build time from the system object library.

### B.1 DISK DRIVER

Valid I/O functions for the disk driver are listed below:

IO.RLB,...,<stadd,size,,blkh,blk>	READ logical block
IO.RPB,...,<stadd,size,,,pbn>	READ physical block
IO.RVB,...,<stadd,size,,blkh,blk>	READ virtual block
IO.WDD,...,<stadd,size,,,pbn>	WRITE physical block (with deleted data mark)
IO.WLB,...,<stadd,size,,blkh,blk>	WRITE logical block
IO.WLC,...,<stadd,size,,blkh,blk>	WRITE logical block followed by write check
IO.WPB,...,<stadd,size,,,pbn>	WRITE physical block
IO.WVB,...,<stadd,size,,blkh,blk>	WRITE virtual block

**B.2 TERMINAL DRIVER**

Valid terminal driver I/O functions are listed below:

IO.ATA, ..., <ast[,parameter2][,ast2]>	ATTACH device, specify unsolicited-character AST <sup>1</sup>
IO.ATT, ...	Attach device
IO.CCO, ..., <stadd,size,vfc>	WRITE logical block, cancel CTRL/O
IO.DET, ...	Detach device
SF.GMC, ..., <stadd,size>	GET multiple characteristics
IO.GTS, ..., <stadd,size>	GET terminal support
IO.KIL, ...	Cancel I/O requests
IO.RAL, ..., <stadd,size[,tmo]>	READ logical block and pass all bits <sup>1</sup>
IO.RLB, ..., <stadd,size[,tmo]>	READ logical block <sup>1</sup>
IO.RNE, ..., <stadd,size[,tmo]>	READ logical block and do not echo <sup>1</sup>
IO.RPR, ..., <stadd,size[,tmo], pradd,prsize,vfc>	READ after prompt <sup>1</sup>
IO.RST, ..., <stadd,size[,tmo]>	READ with special terminators
IO.RTT, ..., <stadd,size[,tmo], table>	READ logical block ended by specified special terminator <sup>2</sup>
IO.RVB, ..., <stadd,size[,tmo]>	READ virtual block <sup>1</sup>
SF.SMC, ..., <stadd,size>	SET multiple characteristics
IO.WAL, ..., <stadd,size,vfc>	WRITE logical block and pass all bits
IO.WBT, ..., <stadd,size,vfc>	WRITE logical block and break through any ongoing I/O
IO.WLB, ..., <stadd,size,vfc>	WRITE logical block
IO.WVB, ..., <stadd,size,vfc>	WRITE virtual block

**B.2.1 Subfunction Bits for Terminal-Driver Functions**

TF.AST	Unsolicited-input-character AST
TF.BIN	Binary prompt
TF.CCO	Cancel CTRL/O
TF.ESQ	Recognize escape sequences
TF.NOT	Unsolicited input AST notification <sup>1</sup>
TF.RAL	Read, pass all bits
TF.RCU	Restore cursor position <sup>1</sup>

---

1. "ast2", "parameter2", and "tmo" parameters are available for full-duplex driver functions only.

2. Full-duplex driver only.

TF.RNE	Read with no echo
TF.RST	Read with special terminators
TF.TMO	Read with time-out <sup>1</sup>
TF.WAL	Write, pass all bits
TF.WBT	Break-through write
TF.XCC	CTRL/C starts a command line interpreter <sup>1</sup>
TF.XOF	Send XOFF

---

1. "asr", "parameter 2", and "tmo" parameters are available for full duplex driver functions only.



# APPENDIX C

## I/O FUNCTION AND STATUS CODES

---

This appendix lists the numeric codes for all I/O functions, directive status returns, and I/O completion status returns. Sections are organized in the following sequence:

- I/O status codes
- Directive status codes
- Device-independent I/O function codes
- Device-dependent I/O function codes

Device-dependent function codes are listed by device. Both devices and codes are organized in alphabetical order.

For each code, the symbolic name is listed in form IO.xxx, IE.xxx, or IS.xxx. A brief description of the error or function is also included. Both decimal and octal values are provided for all codes.

### C.1 I/O STATUS CODES

This section lists error and success codes that can be returned in the I/O status block on completion of an I/O function. The codes may be referenced symbolically by invoking the system macro IOERR\$.

#### C.1.1 I/O Status Error Codes

<i>Name</i>	<i>Decimal</i>	<i>Octal</i>	<i>Meaning</i>
IE.ABO	-15	177761	Operation aborted
IE.ALN	-34	177736	File already open
IE.BAD	-01	177777	Bad parameter

<i>Name</i>	<i>Decimal</i>	<i>Octal</i>	<i>Meaning</i>
IE.BBE	-56	177710	Bad block
IE.BCC	-66	177676	Block check error or framing error
IE.BLK	-20	177754	Illegal block number
IE.BYT	-19	177755	Byte-ligned buffer specified
IE.CNR	-73	177667	Connection rejected
IE.CON	-22	177752	UDC connect error
IE.DAA	-08	177770	Device already attached
IE.DAO	-13	177763	Data overrun
IE.DNA	-07	177771	Device not attached
IE.DNR	-03	177775	Device not ready
IE.DUN	-09	177767	Device not attachable
IE.EOF	-10	177766	End-of-file encountered
IE.EOT	-62	177702	End-of-tape encountered
IE.EOV	-11	177765	End-of-volume encountered
IE.FHE	-59	177705	Fatal hardware error
IE.FLG	-89	177647	Event flag already specified
IE.FLN	-81	177657	ICS/ICR controller already offline
IE.IEF	-97	177637	Invalid event flag
IE.IES	-82	177656	Invalid escape sequence
IE.IFC	-2	177776	Illegal function
IE.MOD	-21	177753	Invalid UDC or ICS/ICR module
IE.NLK	-79	177661	Task not linked to specified ICS/ICR interrupts
IE.NLN	-37	177733	File not open
IE.NOD	-23	177751	No dynamic memory available to allocate a secondary control block
IE.NST	-80	177660	Task specified in ICS/ICR Link or Unlink request not installed
IE.NTR	-87	177651	Task not triggered
IE.OFL	-65	177677	Device off line
IE.ONP	-05	177773	Illegal subfunction
IE.OVR	-18	177756	Illegal read overlay request
IE.PES	-83	177655	Partial escape sequence
IE.PRI	-16	177760	Privilege violation

IE.REJ	-88	177650	Transfer rejected
IE.RSU	-17	177757	Nonsharable resource in use
IE.SPC	-06	177772	Illegal address space
IE.TMO	-74	177666	Time-out error
IE.VER	-04	177774	Unrecoverable error
IE.WCK	-86	177652	Write check error
IE.WLK	-12	177764	Write-locked device

### C.1.2 I/O Status Success Codes

<i>Decimal Name</i>	<i>Octal Bytes</i>	<i>Word</i>	<i>Meaning</i>
IS.CC	Byte 0: 1 Byte 1: 3	001401	Successful completion on read terminated by CTRL/C
IS.CR	Byte 0: 1 Byte 1: 15	006401	Successful completion with RETURN
IS.ESC	Byte 0: 1 Byte 1: 33	015401	Successful completion with ESCape
IS.ESQ	Byte 0: 1 Byte 1: 233	115401	Successful completion with an escape sequence
IS.PND	+00	000000	I/O request pending
IS.RDD	+02	000002	Deleted data mark read
IS.SUC	+01	000001	Successful completion
IS.TMO	+02	000002	Successful completion on read terminated by time-out
IS.TNC	+02	000002	Successful transfer but message truncated (receiver buffer too small)

## C.2 DIRECTIVE CODES

This section lists error and success codes that can be returned in the Directive Status Word at symbolic location \$DSW when a QIO directive is issued.

**C.2.1 Directive Error Codes**

<i>Name</i>	<i>Decimal</i>	<i>Octal</i>	<i>Meaning</i>
IE.ADP	-98	177636	Invalid address
IE.IEF	-97	177637	Invalid event flag number
IE.ILU	-96	177640	Invalid logical unit number
IE.SDP	-99	177635	Invalid DIC number or DPB size
IE.ULN	-05	177773	Unassigned logical unit number
IE.UPN	-01	177777	Insufficient dynamic storage

**C.2.2 Directive Success Codes**

<i>Name</i>	<i>Decimal</i>	<i>Octal</i>	<i>Meaning</i>
IS.SUC	+01	000001	Directive accepted

**C.3 I/O FUNCTION CODES**

This section lists octal codes for all standard and device-dependent I/O functions.

**C.3.1 Standard I/O Function Codes**

<i>Symbolic Name</i>	<i>Word Equivalent</i>	<i>Octal Code (High Byte)</i>	<i>Octal Subcode (Low Byte)</i>	<i>Meaning</i>
IO.ATT	001400	3	0	Attach device
IO.DET	002000	4	0	Detach device
IO.KIL	000012	0	12	Cancel I/O requests
IO.RLB	001000	2	0	Read logical block
IO.RVB	010400	21	0	Read virtual block
IO.WLB	000400	1	0	Write logical block
IO.WVB	011000	22	0	Write virtual block



**C.3.2 Terminal I/O Function Codes**

<i>Symbolic Name</i>	<i>Word Equivalent</i>	<i>Octal Code (High Byte)</i>	<i>Octal Subcode (Low Byte)</i>	<i>Meaning</i>
IO.ATA	001410	3	10	Attach device, specify unsolicited-input-character AST
IO.CCO	000440	1	40	Write logical block and cancel CTRL/O
SF.GMC	002560	5	160	Get multiple characteristics
IO.GTS	002400	5	00	Get terminal support
IO.RAL	001010	2	10	Read logical block and pass all bits
IO.RNE	001020	2	20	Read with no echo
IO.RPR	004400	11	00	Read after prompt
IO.RST	001001	2	1	Read with special terminators
IO.RTT	005001	12	1	Read logical block ended by specified special terminator
SF.SMC	002440	5	40	Set multiple characteristics
IO.WAL	000410	1	10	Write logical block and pass all bits
IO.WBT	000500	1	100	Write logical block and break through on-going I/O

**C.3.3 Subfunction Bits**

With IO.RLB, IO.RPR:

TF.RST	1
TF.BIN	2
TF.RAL	10
TF.RNE	20
TF.XOF	100
TF.TMO	200

With IO.WLB:

TF.WAL	10
TF.CCO	40
TF.WBT	100

With IO.ATT:

TF.AST	10
TF.ESQ	20

## APPENDIX D

# FACILITY AND ERROR CODES

---

The following symbols are used in reporting fatal errors. Their values must not change, since user documentation may refer to them.

### D.1 SUB-FACILITY CODES

These sub-facility codes are included in a fatal error report when one occurs.

<i>Code</i>	<i>Meaning</i>
sfdtbl == 1.	Disable keyboard usage
sfenbl == 2.	Enable keyboard usage
sfmap == 3.	Map keyboard common
sfumap == 4.	Unmap keyboard common
sfinfo == 5.	Determine current keyboard information
sflist == 6.	Generate list of supported keyboards
sfload == 7.	Load specified keyboard
sfmode == 8.	Convert keyboard to specified mode

**D.2 FATAL ERROR CODES**

The following fatal error codes are reported via the subroutine in the diagnostic firmware which reports fatal software-detected errors.

<i>Code</i>	<i>Error</i>
ecipbl = 1. * 1000	Incorrect parameter block length
ecfakc = 2. * 1000	Failure attaching keyboard common (low byte is \$DSW)
ecfcaw = 3. * 1000	Failure creating address window (low byte is \$DSW)
ecfdkc = 4. * 1000	Failure detaching keyboard common (low byte is \$DSW)
ecikrn = 5. * 1000	Invalid keyboard revision number
eccksi = 6. * 1000	Corrupted keyboard-specific information
eckcts = 7. * 1000	Keyboard common too small
ecemti = 8. * 1000	Erroneous mode-table information
ecfcrf = 9. * 1000	Failed to connect RAB to FAB (RMS code in low 9 bits)

**D.3 BUGCHECK MACRO**

This macro defines the following facility and error codes.

<i>Code</i>	<i>Meaning</i>
BF.PKS='B'000100	P/OS Keyboard Handler
BF.TTD='B'000200	Terminal Driver
BF.PTS='B'100400	P/OS Terminal Subsystem
BF.EXE='B'000300	Exec—SSTSR, General
BE.IOT='B'000000	IOT in System State
BE.STK='B'000001	Stack Overflow
BE.BPT='B'000002	Trace Trap or Breakpoint
BE.ILI='B'000003	Illegal Instruction Trap
BE.ODD='B'000004	Odd Address or Other Trap 4
BE.SGF='B'000005	Segment Fault
BE.NPA='B'000006	A Task on P/OS Without a Parent Aborted
BF.UP='B'000400	System Startup Processing
BE.FNF='B'000007	Required File Not Found

# INDEX

---

- \$ Macro form, 3-6
- \$C Macro form, 3-7
- \$S Macro form, 3-7
- .MCALL assembler directive, 10-13
  - arguments, 3-6
  - example, 10-14
  
- ABRT\$ (Abort Task), 9-3
- Access mode, 2-4
  - block, 2-5
  - record, 2-4
- ACS
  - buffer, 8-9
- Active Page Register
  - See APR
- Active task state
  - blocked, 3-17
  - ready-to-run, 3-17
  - stopped, 3-18
- Address mapping, 7-2
- Address space
  - logical, 1-3, 7-2
  - physical, 1-3, 7-2
  - virtual, 1-3, 7-2
- Address window
  - creating, 9-21
- Addressing
  - virtual, 7-2
- Allocate Checkpoint Space
  - See ACS
- ALTP\$ (Alter Priority), 9-5
- ALUN\$ (Assign LUN), 9-7, 10-14
- Application program
  - design suggestions, 1-3
- APR, 7-19
- APRO
  - restriction, 7-19
- Assign LUN
  - See ALUN\$
- AST, 3-3, 5-6
  - characteristics, 5-7
  - disable or inhibit, 9-37
  - service routines, 3-18, 5-8
- ASTOX, 10-6
- ASTX\$\$ (AST Service Exit), 9-9, 10-21
- Asynchronous System Trap
  - See AST
- ATRG\$ (Attach Region), 9-12
  - definition, 7-8
- Attribute list, 8-5
  
- Block access mode
  - sequential, 2-5
  - VBN, 2-5
- Bootblock, 8-15
- Bootstrap, 8-15
  
- Call
  - high-level language, 3-1
- CALL ABORT, 9-3
- CALL ALTPRI, 9-5
- CALL ASNLUN, 9-7
- CALL ATRG, 9-12
- CALL CANALL, 9-28
- CALL CANMT, 9-17
- CALL CLREF, 9-14
- CALL CNCT, 9-19
- CALL CRAW, 9-21
- CALL CRELOG, 9-15
- CALL CRRG, 9-25
- CALL DECLAR, 9-30
- CALL DELLOG, 9-31
- CALL DISCKP, 9-35
- CALL DSASTR, 9-33
- CALL DTRG, 9-36
- CALL ELAW, 9-38
- CALL EMST, 9-40
- CALL ENACKP, 9-43
- CALL ENASTR, 9-42
- CALL EXITIF, 9-44
- CALL EXST, 9-48
- CALL EXTTSK, 9-50
- CALL FEAT, 9-52
- CALL GETDDS, 9-55
- CALL GETLUN, 9-57
- CALL GETMCR, 9-60
- CALL GETPAR, 9-65
- CALL GETREG, 9-67
- CALL GETTIM, 9-69
- CALL GETTSK, 9-71
- CALL GMCX, 9-62
- CALL INASTR, 9-33
- CALL MAP, 9-76
- CALL MARK, 9-76
- CALL QIO, 9-80
- CALL RCST, 9-85
- CALL READEF, 9-92
- CALL RECEIV, 9-87
- CALL RECOEX, 9-89
- CALL REQUES, 9-99
- CALL RESUME, 9-104
- CALL RPOI, 9-96
- CALL RREF, 9-101
- CALL RUN, 9-105
- CALL SDRG, 9-113
- CALL SDRP, 9-116
- CALL SEND, 9-109
- CALL SETDDS, 9-111
- CALL SETEF, 9-119
- CALL SETTIM, 9-137
- CALL SPAWN, 9-123
- CALL SREF, 9-132
- CALL SREX, 9-129
- CALL STLOR, 9-140
- CALL STOP, 9-142
- CALL STOPFR, 9-143
- CALL SUSPND, 9-122
- CALL TRALOG, 9-169
- CALL UNMAP, 9-150
- CALL USTP, 9-152
- CALL VRCD, 9-153

- CALL VRCS, 9-155
- CALL VRCX, 9-157
- CALL VSDA, 9-159
- CALL VSRC, 9-161
- CALL WAITFR, 9-171
- CALL WFLOR, 9-169
- CALL WFSNE, 9-167
- CALL WIMP, 9-163
- CALL WTQIO, 9-83
- Callable system routines, 8-1
  - general conventions, 8-1
- Callable task routines, 8-3
- Cancel Mark Time
  - See CMKT\$
- Cancel Time Based Requests
  - See CSRQ\$
- CBD, 9-25
- Checkpointing
  - affected task states, 1-5
  - definition, 1-5
  - disabled, 9-35
  - enable, 9-43
- CLEF\$ (Clear Event Flag), 9-14
- CLOG\$ (Create Logical Name String), 4-2, 9-15
  - example, 4-2
- CMKT\$ (Cancel Mark Time), 9-17
- CNCT\$ (Connect), 9-19
- Command line
  - passing, 9-96
- Common Block Directory
  - See CBD
- Common event flag
  - definition, 5-2
- Communications Driver, 13-1
- Configuration table, 9-163
- Connect, 6-2
  - See CNCT\$
- CRAW\$ (Create Address Window), 7-8, 9-21
- Create Address Window
  - See CRAW\$
- Create Logical Name
  - See CLOG\$, PROLOG
- Create Region
  - See CRRG\$
- CREDEL
  - server task, 8-6
- CRRG\$ (Create Region), 9-25
  - definition, 7-8
- CSRQ\$ (Cancel Time Based Requests), 9-28
  
- DECL\$\$ (Declare Significant Event), 9-30
- Default directories, 4-3
- Default directory string
  - retrieving, 4-5
  - setting up, 4-4
- Delete Logical Name
  - See DLOG\$, PROLOG
- Detach Region
  - See DTRG\$
- Device
  - physical device names, 10-15
  - pseudo-device names, 10-15
  - standard devices, 11-1
  - supported devices, 10-2
  
- DIC, 3-2
- DIR\$ Macro, 10-13
- Directive
  - conventions, 3-24
  - description format, 9-1
  - event-associated, 3-24
  - informational, 3-21
  - memory management, 3-23
  - parent/offspring tasking, 3-22
  - task status control directives, 3-21
  - trap-associated, 3-22
- Directive Identification Code
  - See DIC
- Directive macros, 3-4
- Directive Parameter Block
  - See DPB
- Directive Status Word
  - See DSW
- Directory
  - creating a, 8-6
  - deleting a, 8-6
  - setting up default, 9-123
- Directory manipulation
  - See PRODIR
- Disable AST Recognition
  - See DHAR\$
- Disk driver
  - I/O functions, 8-1
- DLOG\$ (Delete Logical Name), 4-3, 9-31
  - example, 4-4
- DPB, 3-2
  - created at assembly time, 3-6
  - created at run time, 3-7
  - creation of, 3-4
  - definition, 10-10
- Driver, Communications, 13-1
- DSAR\$\$ (Disable AST Recognition), 9-33
- DSCP\$\$ (Disable Checkpointing), 9-35
- DSW, 3-2
- DTRG\$ (Detach Region), 7-8, 9-36
- Dynamic region, 7-3
  
- EFN, 5-2
- ELAW\$ (Eliminate Address Window), 7-8, 9-38
- EMST\$ (Emit Status), 6-3, 9-40
- EMT 377 instruction, 3-1
- ENAR\$\$ (Enable AST Recognition), 9-42
- ENCP\$\$ (Enable Checkpointing), 9-43
- Equivalence name, 4-1
- Error codes, A-1, D-1
- Error returns, 3-3
- Error routine address, 3-8
- Event flag
  - definition, 5-2
  - setting, 9-119
  - testing for, 5-4
- Event Flag Number
  - See EFN
- EXIF\$ (Exit If), 9-44
- Exit With Status directive
  - See EXST\$
- EXIT\$\$ (Task Exit), 9-46
- EXST\$ (Exit With Status), 9-48
- EXTK\$ (Extend Task), 9-50

- FCP
  - See Files-11 ACP
- FEAT\$ (Test Extended Feature), 9-52
- File
  - access modes, 2-4
  - accessing file attributes, 8-4
  - directory manipulation, 8-6
  - identification block, 8-4
  - indexed, 2-4
  - list of accessible attributes, 8-5
  - organization, 2-3
  - record formats, 2-3
  - relative, 2-4
  - sequential, 2-3
  - structure, 2-3
- File system, 2-1
  - data storage, 2-1
  - overview, 2-1
  - See also RMS
- Files-11 ACP (FCP)
  - description, 2-2
  - logical name use, 4-3
- Floating Point Processor
  - exception ASTs, 9-120
- Fortran, 3-1
  - use of AST service routines,
- Fortran Object Time System
  - See DTS
- Fortran subroutines
  - calls, 3-12
  - corresponding macro calls, 3-12
  - error conditions, 3-15
  - GETADR subroutine, 3-12
  - integer arguments, 3-11
  - optional arguments, 3-11
  - system directive operations, 3-10
  - use of, 3-10
  
- GDIR\$ (Get Default Directory), 4-7, 9-55
  - example, 4-7
- GET file attributes
  - function of PROATR, 8-4
- Get Mapping Context
  - See GMCX\$
- Get Partition Parameters
  - See GPRT\$
- Get Region Parameters
  - See GREG\$
- Get Task Parameters
  - See GTSK\$
- Get Time Parameters
  - See GTIM\$
- Global symbols, 3-8
- GLUN\$ (Get LUN Info), 9-57, 10-18
- GMCR\$ (Get Command Line), 9-60
- GMCX\$ (Get Mapping Context), 7-9, 9-62
- GPRT\$ (Get Partition Parameters), 9-65
- GREG\$ (Get Region Parameters), 7-9, 9-67
- GTIM\$ (Get Time), 9-69
- GTSK\$ (Get Task Parameters), 9-71
  
- I/O
  - attaching devices
    - See IO.ATT
  - canceling requests
    - See IO.KIL
  - detaching devices
    - See IO.DET
  - general functions, 10-1
  - logical, 10-2
  - physical, 10-2
  - standard functions, 10-19
  - virtual, 10-2
- I/O completion
  - Executive actions, 10-28
- I/O request
  - acceptance of, 10-5
  - issuing, 10-4
  - rejection of, 10-5
- IHAR\$\$ (Inhibit AST Recognition), 9-33
- Initialization
  - volume, 8-7
- Instrument Society of America
  - See ISA
- Integer array, 3-11
- Intertask synchronization
  - examples, 6-4
- IO.ATT, 10-5, 10-20
- IO.DET, 10-21
- IO.KIL, 10-21
- IO.RVB, 10-22
- IO.WVB, 10-23
- ISA
  - and AST service routines, 9-9
  - Fortran calls, 3-3
  
- LBN (Logical Block Number), 2-2
- Library
  - cluster, 1-4
  - POSSUM, 8-1
  - shared, 1-4
- Local event flag
  - definition, 5-2
  - examples of use, 5-3
- Local symbolic offset, 3-10
- Logical address space, 7-2
- Logical Block Number
  - See LBN
- Logical name
  - create, 4-3
  - definition, 4-1
  - delete, 4-3, 9-31
  - duplicate, 4-2
  - Files-11 use, 4-3
  - logical name table, 4-1
  - RMS conventions, 4-2
  - RMS translation, 4-2
- Logical Unit Table
- LUN (Logical Unit Number)
  - changing the assignment, 10-4
  - definition, 10-3
  - reassignment, 10-3
- LUT (Logical Unit Table), 10-4

- Macro call
  - examples, 3-9
- Macro expansion
  - \$ form, 3-6
  - \$C form, 3-7
  - \$S form, 3-7
- Macro name conventions, 3-6
- MACRO-11, 3-1
  - use of system directives, 3-1
- MAP\$ (Map Address Window), 7-9, 9-73
- Mapping, 7-2
- Mark Time
  - See MRKT\$
- Memory common
  - fixing in memory, 8-18
  - installation of, 8-16
  - removal of, 8-18
- Memory Management directives, 7-1
- MRKT\$ (Mark Time), 9-76
  
- OCB (Offspring Control Block), 6-2, 9-19
- Offspring Control Block
  - See OCB
- Offspring task, 6-1
  - exit status, 6-3
- OTS
  - diagnostic messages, 3-18
- Overlay
  - disk-resident, 1-4
  - memory-resident, 1-4
  
- Parent task, 6-1
- Parent/offspring tasking
  - chaining, 6-1
  - definition, 6-1
  - spawning, 6-1
  - use of, 6-1
- Partition Control Block
  - See PCB
- PC, 5-5
- PCB, 9-30
- PDP-11 R5 Calling Sequence
  - for high level-level languages, 8-2
- Physical Address Space, 7-2
- POSSUM library, 8-1
  - included in a task, 8-1
  - linking a task to POSSUM, 8-1
- Privileged tasks
  - remapping APRs to regions, 7-19
- PROATR, 8-4
  - arguments, 8-4
- Processor Status
  - See PS
- Processor Status Word
  - See PSW
- PRODIR, 8-6
  - arguments, 8-6
- PROFBI, 8-7
  - arguments, 8-7
- Program Counter
  - See PC
- PROLOG, 8-11
- PROVOL, 8-14
  - arguments, 8-15
- PS, 5-5
- PSW, 3-3
  
- QIO
  - macro expansion, 10-5
  - macro format, 10-6
  - standard functions for disks, 11-3
  - typical parameters, 10-5
- QIO\$ (Queue I/O Request), 9-80, 10-14
- QIOW\$, 9-83, 10-13
  
- R5 calling sequence, 8-2
- RCST\$ (Receive Data or Stop), 9-85
- RCVD\$ (Receive Data), 9-87
- RCVX\$ (Receive Data or Exit), 9-89
- RD50
  - description, 11-2
- RDAF\$ (Read All Event Flags), 9-92
- RDB, 7-10
  - definition, 7-10
  - field values, 7-18
  - generating with Fortran, 7-13
  - generating with macros, 7-13
- RDB array format, 7-14
- RDBBK\$, 7-12
- RDBDF\$, 7-12
- RDEF\$ (Read Event Flag), 9-93
- RDXF\$ (Read Extended Event Flags), 9-94
- Read All Event Flags
  - See RDAF\$
- Read Event Flag
  - See RDEF\$
- Read Extended Event Flags
  - See RDXF\$
- Receive By Reference
  - See RREF\$
- Receive Data
  - See RCVD\$
- Receive Data Or Exit
  - See RCVX\$
- Receive Data Or Stop
  - See RCST\$
- Record access mode
  - key, 2-4
  - record file access (RFA), 2-4
  - sequential, 2-5
- Record formats
  - fixed length, 2-3
  - stream, 2-3
  - undefined, 2-3
  - variable length with VFC, 2-3
- Record Management System
  - See RMS
- Region, 9-12
  - attaching to, 7-7
  - creation, 9-28
  - definition, 7-3
  - dynamic, 7-3
  - fixing in memory, 8-18
  - ID, 7-4
  - installation of, 8-16
  - protecting, 7-7
  - shareable, 7-7
  - static common region, 7-3
- Region Definition Block
  - See RDB
- Request
  - issuing, 9-99
- Request Task
  - See RQST\$
- Resume Task
  - See RSUM\$



- RMS
  - and default directories, 4-3
  - associated documents, 2-5
  - data storage, 2-1
  - overview, 2-1
- RPOI\$, 9-96
  - and Spawn directive, 6-2
  - when to use, 1-3
- RQST\$
  - when to use, 1-3
- RQST\$ (Request Task), 9-99
- RREF\$
  - definition, 7-9
- RREF\$ (Receive By Reference), 9-101
- RSUM\$ (Resume Task), 9-104
- RSXMAC.SML
  - See System macro library
- RUN\$ (Run Task), 9-105
- RX50, 11-2
  
- SDAT\$ (Send Data), 9-109
- SDIR\$ (Setup Default Directory), 4-6, 9-111
  - example, 4-6
- SDRC\$ (Send, Request and Connect), 9-113
  - when to use, 1-3
- SDRP\$, 9-116
- Send By Reference
  - See SREF\$
- Send Data
  - See SDAT\$
- Send, Request and Connect, 6-2
  - See SDRC\$
- Set Event Flag
  - See SETF\$
- SET file attributes
  - function of PROATR, 8-4
- Set System Time
  - See STIM\$
- SETF\$ (Set Event Flag), 9-119
- Setup Default Directory String
  - See SDIR\$
- SFPA\$, 9-120
- Shareable region, 7-3
- Shared regions, 7-7
- Significant event, 9-101, 10-8
  - declaration, 9-30
  - definition, 5-1
  - example, 10-9
  - wait for, 9-169
- Spawn
  - See SPWN\$
- Spawning, 6-1
- Specify Receive Data AST
  - See SRDA\$
- Specify Requested Exit AST
  - See SREX\$
- SPND\$\$S (Suspend), 9-122
- SPWN\$ (Spawn), 9-123
  - when to use, 1-4
- SRDA\$ (Specify Receive Data AST), 9-127
- SREF\$
  - definition, 7-11
- SREF\$ (Send By Reference), 9-132
- SREX\$ (Specify Requested Exit AST), 9-129
- SRRAS\$, 9-135
  
- SST, 3-22, 5-4
  - definition, 5-4
  - service routines, 5-5
  - vector table, 5-5
  - vector table format, 5-5
- Stack Pointer, 3-7
- Static common region
  - definition, 7-5
- Status control block
  - format, 8-3
- STD, 3-16
- STIM\$ (Set System Time), 9-137
- STLO\$, 9-140
- Stop
  - See STOP\$\$S
- Stop For Single Event Flag
  - See STSE\$
- STOP\$\$S, 9-142
- Stop-bit synchronization, 5-11
- STSE\$, 9-143
- Suspend
  - See SPND\$\$S
- SVDB\$, 9-144
- SVTK\$, 9-146
- SWST\$ (Switch State), 9-148
- Synchronous System Trap
  - See SST
- System directive, 3-1
  - definition, 3-1
  - processing, 3-2
- System library account
  - system macros, 10-5
- System Macro Library, 3-1
  - RSXMAC.SML, 3-6
- System object module library, 3-1
- System Task Directory
  - See STD
- System trap, 5-4
  
- Task
  - addressing capability, 7-1
  - callable task routines, 8-4
  - changing priority, 9-5
  - cooperating tasks, 1-4
  - extending size of, 9-50
  - offspring task, 6-1
  - overlying, 7-1
  - parent task, 6-1
  - resuming suspended, 9-104
  - server task, 8-1
  - spawning, 6-1, 9-123
  - stopping, 5-10, 9-142
  - suspension of, 9-122
  - unstopping, 5-10, 9-153
- Task Communication, 6-3
- Task Control Block
  - See TCB
- Task names
  - defining, 3-11
  - length, 3-11
- Task region, 7-3

- Task state, 3-16
  - active, 3-16
  - dormant, 3-16
- Task state transitions
  - active to dormant, 3-18
  - blocked to ready-to-run, 3-18
  - blocked to stopped, 3-18
  - dormant to active, 3-17
  - ready-to-run to blocked, 3-21
  - ready-to-run to stopped, 3-17
  - stopped to blocked, 3-18
  - stopped to ready-to-run, 3-18
- TCB, 5-7
- Terminal driver
  - features, 12-1
  - QIO macro functions for, 12-3
  - subfunction bits, B-2
  - valid I/O functions, B-2
- Test Extended Feature
  - See FEAT\$
- Tick
  - definition, 9-87
- Trap
  - asynchronous, 10-9
  - synchronous, 10-9
  - system, 10-9
  
- UIC, 9-78
- UMAP\$ (Unmap Address Window), 7-9, 9-150
- User data structures, 7-9
- User Identification Code
  - See UIC
- USTP\$ (Unstop Task), 9-152
  
- Variable Receive Data
  - See VRCD\$
- Variable Receive Data Or Exit
  - See VRCX\$
- Variable Receive Data Or Stop
  - See VRCS\$
- Variable Send Data
  - See VSDA\$
- Variable Send, Request and Connect
  - See VSRC\$
- VBN (Virtual Block Number), 2-2
- Virtual address space, 7-2
- Virtual address window
  - definition, 7-2
- Virtual block
  - reading, 10-22
  - writing, 10-23
- Virtual Block Number
  - See VBN
- Volume
  - bad block checking, 8-7
  - bootstrap, 8-15
  - dismounting, 8-15
  - foreign, 8-15
  - formatting, 8-7
  - initialization, 8-7
  - label, 8-7
  - mounting, 8-15
  - write bootblock, 8-15
- VRCD\$ (Variable Receive Data), 9-153
- VRCS\$, 9-155
- VRCX\$, 9-157
- VSDA\$ (Variable Send Data), 9-159
- VSRC\$, 9-161
  - when to use, 1-3
  
- Wait For Significant Event
  - See WSIG\$
- Wait For Single Event Flag
  - See WTSE\$
- WDB, 7-15
  - field values, 7-18
  - generating with Fortran, 7-17
- WDBBK\$, 7-16
- WDBDF\$, 7-16
- WIMP\$, 9-187
- Window Definition Block
  - See WDB
- WSIG\$, 9-167
- WTLO\$, 9-169
- WTSE\$, 9-171, 10-21

- Task Communication, 6-2
- Task Control Block
  - See TCB
- Task names
  - defining, 3-11
  - length, 3-11
- Task naming
  - in Executive-level dispatching, 3-22
- Task region, 7-3
- Task state, 3-16
  - active, 3-16
  - dormant, 3-16
- Task state transitions
  - active to dormant, 3-18
  - blocked to ready-to-run, 3-18
  - blocked to stopped, 3-18
  - dormant to active, 3-17
  - ready-to-run to blocked, 3-17
  - ready-to-run to stopped, 3-17
  - stopped to blocked, 3-18
  - stopped to ready-to-run, 3-18
- TCB, 5-7, 5-10
- Terminal driver
  - features, 12-1
  - QIO macro functions for, 12-3
  - subfunction bits, B-2
  - valid I/O functions, B-2
- Test Extended Feature
  - See FEAT\$
- Tick
  - definition, 9-78
- TLOG\$ (Translate Logical Name), 4-3, 9-149
  - example, 4-3
- Translate Logical Name
  - See TLOG\$, PROLOG
- Trap
  - asynchronous, 10-9
  - synchronous, 10-9
  - system, 10-9
- UIC, 9-71
- UMAP\$ (Unmap Address Window), 7-9, 9-150
- User data structures, 7-9
- User Identification Code
  - See UIC
- USTP\$ (Unstop Task), 9-152
- Variable Receive Data
  - See VRCD\$
- Variable Receive Data Or Exit
  - See VRCX\$
- Variable Receive Data Or Stop
  - See VRCS\$
- Variable Send Data
  - See VSDA\$
- Variable Send, Request and Connect
  - See VSRC\$
- VCN (Virtual Block Number), 2-2
- Virtual address space, 7-2
- Virtual address window
  - definition, 7-2
- Virtual block
  - reading, 10-22
  - writing, 10-23
- Virtual Block Number
  - See VBN
- Volume
  - bad block checking, 8-7
  - bootstrap, 8-23
  - dismounting, 8-22
  - foreign, 8-23
  - formatting, 8-7
  - initialization, 8-7
  - label, 8-8
  - mounting, 8-22
  - write bootblock, 8-23
- VRCD\$ (Variable Receive Data), 9-153
- VRCS\$, 9-155
- VRCX\$, 9-157
- VSDA\$ (Variable Send Data), 9-159
- VSRC\$, 9-161
  - when to use, 1-3
- Wait For Significant Event
  - See WSIG\$
- Wait For Single Event Flag
  - See WTSE\$
- WDB, 7-10, 7-15
  - field values, 7-18
  - generating with Fortran, 7-17
- WDBBK\$, 7-16
- WDBDF\$, 7-16
- WIMP\$, 9-163
- Window Definition Block
  - See WDB
- WSIG\$, 9-167
- WTLO\$, 9-169
- WTSE\$, 9-171, 10-18
- XK communications driver, 13-1
  - QIO macro functions for, 13-2



**READER'S COMMENTS**

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.

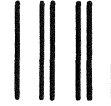
- Assembly language programmer
- Higher-level language programmer
- Occasional programmer (experienced)
- User with little programming experience
- Student programmer
- Other (please specify) \_\_\_\_\_

Please cut along this line.

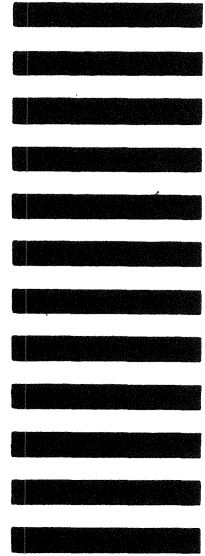
Name \_\_\_\_\_ Date \_\_\_\_\_  
Organization \_\_\_\_\_  
Street \_\_\_\_\_  
City \_\_\_\_\_ State \_\_\_\_\_ Zip Code \_\_\_\_\_  
or  
Country \_\_\_\_\_

----- Do Not Tear - Fold Here and Tape -----

**digital**



No Postage  
Necessary  
if Mailed in the  
United States



**BUSINESS REPLY MAIL**  
FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

Professional 300 Series Publications  
DIGITAL EQUIPMENT CORPORATION  
146 MAIN STREET  
MAYNARD, MASSACHUSETTS 01754

----- Do Not Tear - Fold Here -----

Cut Along Dotted Line