VMS Obsolete Features Manual

Order Number: AA-LB25A-TE

April 1988

This manual contains information about DCL commands and qualifiers, system services, RTL routines, and VMS utilities and components that are now obsolete. The manual also includes an appendix of DCL commands and qualifiers, RTL routines, and VMS utilities and components that have been eliminated from VMS.

Revision/Update Information:	This is a new manual.
Software Version:	VMS Version 5.0

digital equipment corporation
maynard, massachusetts

April 1988

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

The following are trademarks of Digital Equipment Corporation:

DEC	DIBOL	UNIBUS
DEC/CMS	EduSystem	VAX
DEC/MMS	IAS	VAXcluster
DECnet	MASSBUS	VMS
DECsystem–10	PDP	VT
DECSYSTEM–20	PDT	
DECUS	RSTS	
DECwriter	RSX	digital™

ZK-4710

HOW TO ORDER ADDITIONAL DOCUMENTATION
DIRECT MAIL ORDERS

USA*

Digital Equipment Corporation
P.O. Box CS2008
Nashua, New Hampshire
03061

CANADA

Digital Equipment
of Canada Ltd.
100 Herzberg Road
Kanata, Ontario K2K 2A6
Attn: Direct Order Desk

INTERNATIONAL

Digital Equipment Corporation
PSG Business Manager
c/o Digital's local subsidiary
or approved distributor

In Continental USA, Alaska, and Hawaii call 800-DIGITAL.

In Canada call 800-267-6215.

*Any order from Puerto Rico must be placed with the local Digital subsidiary (809-754-7575).

Internal orders should be placed through the Software Distribution Center (SDC), Digital Equipment Corporation, Westminster, Massachusetts 01473.

Production Note

This book was produced with the VAX DOCUMENT electronic publishing system, a software tool developed and sold by DIGITAL. In this system, writers use an ASCII text editor to create source files containing text and English-like code; this code labels the structural elements of the document, such as chapters, paragraphs, and tables. The VAX DOCUMENT software, which runs on the VMS operating system, interprets the code to format the text, generate a table of contents and index, and paginate the entire document. Writers can print the document on the terminal or line printer, or they can use DIGITAL-supported devices, such as the LN03 laser printer and PostScript® printers (PrintServer 40 or LN03R ScriptPrinter), to produce a typeset-quality copy containing integrated graphics.

® PostScript is a registered trademark of Adobe Systems, Inc.

Contents

PREFACE ix

CHAPTER 1 OBSOLETE DCL COMMANDS AND QUALIFIERS 1–1

1.1 OBSOLETE DCL COMMANDS 1–1
SET CLUSTER/QUORUM 1–2
SET DEVICE/ACL 1–4
SET DIRECTORY/ACL 1–8
SET FILE/ACL 1–14
SET QUEUE/ENTRY 1–20

1.2 OBSOLETE DCL COMMAND QUALIFIERS 1–29
1.2.1 Obsolete Qualifiers of the INITIALIZE/QUEUE Command 1–29
1.2.1.1 INITIALIZE/QUEUE/[NO]BURST • 1–29
1.2.1.2 INITIALIZE/QUEUE/[NO]FLAG • 1–30
1.2.1.3 INITIALIZE/QUEUE/PRIORITY • 1–30
1.2.1.4 INITIALIZE/QUEUE/TERMINAL • 1–30
1.2.2 Obsolete Qualifiers of the START/QUEUE Command 1–30
1.2.2.1 START/QUEUE/[NO]BATCH • 1–30
1.2.2.2 START/QUEUE/PRIORITY=n • 1–31
1.2.2.3 START/QUEUE/TERMINAL • 1–31

CHAPTER 2 OBSOLETE SYSTEM SERVICES 2–1
$BRDCST 2–2
$CNTREG 2–6
$CRELOG 2–8
$DELLOG 2–10
$GETCHN 2–12
$GETDEV 2–16
$INPUT 2–19
$OUTPUT 2–20
$SNDACC 2–22
$SNDSMB 2–26
$TRNLOG 2–42

CHAPTER 3 OBSOLETE RTL ROUTINES 3–1

3.1 OBSOLETE FOR$ ROUTINES 3–1

3.2 OBSOLETE TERMINAL-INDEPENDENT SCREEN MANIPULATION PROCEDURES 3–2
3.2.1 Obtaining Screen Information 3–3
3.2.2 Positioning the Cursor on the Screen 3–4
3.2.2.1 Controlling Input from and Output to the Screen • 3–5
3.2.2.2 Buffering Screen I/O • 3–7
3.2.2.3 Using Screen Procedures with Files and Hardcopy Terminals • 3–8

3.3 LIB$EMULATE 3–9

3.4 OBSOLETE RTL ROUTINES 3–9
FOR$CNV_OUT_I 3–10
FOR$CNV_OUT_L 3–11
FOR$CNV_OUT_O 3–12
FOR$CNV_OUT_Z 3–13
FOR$CNV_IN_I 3–14
FOR$CNV_IN_L 3–16
FOR$CNV_IN_O 3–18
FOR$CNV_IN_DEFG 3–20
FOR$CNV_IN_Z 3–23
LIB$DOWN_SCROLL 3–25
LIB$EMULATE 3–26
LIB$ERASE_LINE 3–28
LIB$ERASE_PAGE 3–30
LIB$GET_SCREEN 3–31
LIB$PUT_BUFFER 3–33
LIB$PUT_LINE 3–35
LIB$PUT_SCREEN 3–37
LIB$SCREEN_INFO 3–39
LIB$SET_BUFFER 3–41
LIB$SET_CURSOR 3–43
LIB$SET_OUTPUT 3–45
LIB$SET_SCROLL 3–48
LIB$STOP_OUTPUT 3–49
LIB$UP_SCROLL 3–50
SCR$DOWN_SCROLL 3–51
SCR$ERASE_LINE 3–52

SCR$ERASE_PAGE 3–53
SCR$GET_SCREEN 3–54
SCR$SCREEN_INFO 3–56
SCR$PUT_BUFFER 3–57
SCR$PUT_LINE 3–58
SCR$PUT_SCREEN 3–60
SCR$SET_BUFFER 3–62
SCR$SET_CURSOR 3–64
SCR$SET_OUTPUT 3–65
SCR$SET_SCROLL 3–68
SCR$STOP_OUTPUT 3–69
SCR$UP_SCROLL 3–70

CHAPTER 4 OBSOLETE UTILITIES AND UTILITY COMPONENTS 4–1

4.1 OBSOLETE UTILITIES 4–1
DISK QUOTA 4–2
4.1.1 Establishing Disk Quotas 4–2
4.1.2 Creating a Quota File 4–2
4.1.3 Maintaining a Quota File 4–3
4.1.4 Disabling a Quota File 4–3
4.1.5 Listing of Commands 4–3

DISKQUOTA Commands 4–5
ADD 4–6
CREATE 4–7
DISABLE 4–8
ENABLE 4–9
EXIT 4–10
HELP 4–11
MODIFY 4–12
REBUILD 4–13
REMOVE 4–14
SHOW 4–15
USE 4–16

4.2 OBSOLETE COMPONENTS OF CURRENT VMS UTILITIES 4–17

APPENDIX A ELIMINATED FEATURES A–1
ELIMINATED FEATURES A–2

TABLES

1–1 Table of Obsolete DCL Commands 1–1
1–2 Table of Obsolete DCL Command Qualifiers 1–29
2–1 Table of Obsolete System Services 2–1
3–1 Table of Obsolete RTL Routines 3–1
3–2 The Terminal-Independent Screen Procedures 3–3
3–3 Screen Attributes 3–6
4–1 DISKQUOTA Command Summary 4–3

Preface

Intended Audience

This manual is for those users who maintain programs and command procedures containing references or calls to the now obsolete DCL commands, system services, and RTL routines as well as for all those who use the now obsolete utilities and commands.

This manual is new for VMS Version 5.0. However, its contents have appeared previously in earlier versions of the VMS manuals that describe the individual components.

Structure of the Document

This manual is organized by VMS operating system components, with one chapter each for DCL commands, system services, RTL routines, and utilities.

The contents of the chapters are as follows:

- Chapter 1 lists obsolete DCL commands and their replacements. The chapter also contains the DCL Dictionary entry for each obsolete command.
- Chapter 2 lists obsolete system services and their replacements. The chapter also describes each obsolete system service.
- Chapter 3 lists obsolete Run-Time Library routines and their replacements. The chapter also describes each obsolete routine.
- Chapter 4 describes obsolete VMS utilities and their replacements. The chapter also lists and describes obsolete components of current VMS utilities.

The manual also contains an appendix that describes DCL commands, RTL routines, and utilities and utility components eliminated from VMS. Note that eliminated features no longer work and do not have replacements.

Associated Documents

This manual describes obsolete features of VMS. For detailed reference information about replacement features, see *VMS DCL Dictionary, VMS Run-Time Library Routines Volume, VMS System Services Volume,* or *VMS SYSMAN Utility Manual.*

Conventions

Convention	Meaning
RET	In examples, a key name (usually abbreviated) shown within a box indicates that you press a key on the keyboard; in text, a key name is not enclosed in a box. In this example, the key is the RETURN key. (Note that the RETURN key is not usually shown in syntax statements or in all examples; however, assume that you must press the RETURN key after entering a command or responding to a prompt.)
CTRL/C	A key combination, shown in uppercase with a slash separating two key names, indicates that you hold down the first key while you press the second key. For example, the key combination CTRL/C indicates that you hold down the key labeled CTRL while you press the key labeled C. In examples, a key combination is enclosed in a box.
$ SHOW TIME 05-JUN-1988 11:55:22	In examples, system output (what the system displays) is shown in black. User input (what you enter) is shown in red.
$ TYPE MYFILE.DAT . . .	In examples, a vertical series of periods, or ellipsis, means either that not all the data that the system would display in response to a command is shown or that not all the data a user would enter is shown.
input-file, . . .	In examples, a horizontal ellipsis indicates that additional parameters, values, or other information can be entered, that preceding items can be repeated one or more times, or that optional arguments in a statement have been omitted.
[logical-name]	Brackets indicate that the enclosed item is optional. (Brackets are not, however, optional in the syntax of a directory name in a file specification or in the syntax of a substring specification in an assignment statement.)
quotation marks apostrophes	The term quotation marks is used to refer to double quotation marks ("). The term apostrophe (') is used to refer to a single quotation mark.

New and Changed Features

This is a new manual for VMS Version 5.0.

1 Obsolete DCL Commands and Qualifiers

This chapter describes all DCL commands and qualifiers that have been made obsolete since VMS Version 4.0. Obsolete commands and obsolete qualifiers have been superseded by more flexible or more efficient commands and qualifiers and are no longer updated. DIGITAL recommends that you use current commands when you write new programs. Old programs that use obsolete commands and qualifiers should be updated with current commands and qualifiers.

1.1 Obsolete DCL Commands

This section describes obsolete DCL commands. Table 1–1 lists the obsolete command in the left column, the version of VMS that superseded the obsolete command in the center column, and the current command in the third column. For example, SET CLUSTER/QUORUM was replaced in VMS Version 5.0 by SET CLUSTER/EXPECTED_VOTES.

Table 1–1 Table of Obsolete DCL Commands

Obsolete Command	Made Obsolete in Version	Current Command
SET CLUSTER/QUORUM	5.0	SET CLUSTER/EXPECTED_VOTES
SET DEVICE/ACL	5.0	SET ACL/OBJECT_TYPE=DEVICE
SET DIRECTORY/ACL	5.0	SET ACL/OBJECT_TYPE=FILE
SET FILE/ACL	5.0	SET ACL/OBJECT_TYPE=FILE
SET QUEUE/ENTRY	5.0	SET ENTRY

SET CLUSTER/QUORUM

Sets the cluster quorum to a value that you specify or, if no value is specified, sets the cluster quorum to a value determined by the system. The /QUORUM qualifier is required.

In VMS Version 5.0, the SET CLUSTER/QUORUM command was replaced by SET CLUSTER/EXPECTED_VOTES. For more information on the replacement command see the Version 5.0 *VMS DCL Dictionary*.

FORMAT

SET CLUSTER/QUORUM[=quorum-value]

RESTRICTIONS

Requires operator (OPER) privilege.

PARAMETERS

None.

DESCRIPTION

The SET CLUSTER/QUORUM command enables you to manually adjust the cluster quorum value. The cluster quorum is based upon an estimate of the total number of votes available in the cluster. It is automatically adjusted upward as new systems join.

You can specify the desired quorum value as part of the SET CLUSTER /QUORUM command string. If you issue the SET CLUSTER/QUORUM command without specifying a value for quorum, the system calculates the value for you, using the formula:

```
(V+2)/2
```

where V is the sum of the votes of all nodes that are currently in the cluster. The system will not allow you to set the quorum to a value less than or equal to the value calculated by the system formula or to a value greater than the number of votes present.

When you issue the SET CLUSTER/QUORUM command without specifying a quorum value, the system assumes that all nodes that are expected to be in the cluster are currently members.

In general, you use the SET CLUSTER/QUORUM command only when a node is leaving the cluster for an extended period of time. Under normal circumstances, quorum is not reduced when a node leaves the cluster, since it is assumed that the node may be rebooted and rejoin the cluster. If a node is removed from the cluster and is unable to rejoin the cluster within a reasonable period of time (for example, if a node crashes due to a hardware problem and cannot rejoin the cluster for several days), the quorum for the cluster can safely be reduced until that node rejoins.

The purpose of the quorum is to eliminate any possibility of the cluster partitioning into separate clusters and simultaneously accessing the same resources (such as HSC50 disks). If the sum of the votes of all members of the cluster is smaller than the cluster quorum, all nodes in the cluster will block activity until new nodes join to increase the vote total. Lowering

the quorum value when one or more nodes leave the cluster, reduces the possibility of this happening.

When you issue the SET CLUSTER/QUORUM command, either with or without a quorum value specified, the system will respond with a message indicating the new quorum value that was actually set. Note that you need only issue this command on one node in the cluster, since the new value for the quorum will be propogated through the cluster. This new quorum value should then be stored in the SYSGEN parameter QUORUM on each cluster node, so that it remains in effect after the nodes reboot.

When a node that was previously a member of the cluster is ready to rejoin, you should increase the SYSGEN parameter QUORUM to its original value before bringing the node back to the cluster. Note that you do not need to use the SET CLUSTER/QUORUM command to increase the cluster quorum, since the quorum value will be increased automatically when the node rejoins the cluster.

EXAMPLES

1
```
$ SET CLUSTER/QUORUM
```

This command instructs the system to calculate the cluster quorum value for you, since no value is specified as part of the command string. The system will use the formula: (V+2)/2, described above.

2
```
$ SET CLUSTER/QUORUM=9
```

This command sets the cluster quorum to 9, which is the value specified in the command string.

SET DEVICE/ACL

Allows you to modify the access control list (ACL) of a device. The /ACL qualifier is required.

In VMS Version 5.0, the SET DEVICE/ACL command was replaced by SET ACL/OBJECT_TYPE=DEVICE. For more information on the replacement command see the Version 5.0 *VMS DCL Dictionary*.

FORMAT

SET DEVICE/ACL*[=(ace[,...])] device-name*

RESTRICTIONS

None.

PARAMETERS

device-name

Specifies a device whose access control list (ACL) is being modified. Wildcard characters are not allowed in the device name.

(ace[,...])

Specifies one or more access control entries (ACEs) to be modified. When no ACE is specified, the entire access control list is affected. Separate multiple ACEs with commas. The specified ACEs are inserted at the top of the ACL unless the /AFTER qualifier is specified.

DESCRIPTION

The SET DEVICE/ACL command enables you to manipulate the entire access control list (ACL) of a device, or to create, modify, or delete access control entries (ACEs) in the ACL of a device. The only valid access to be specified in an ACE for a device is READ access, which means that the device can be allocated.

To use the SET DEVICE/ACL command, you specify the name of the device whose ACL you want to manipulate.

The SET DEVICE/ACL command can be used to add ACEs to an ACL. For example, the following command adds an ACE to the ACL of the terminal device TTA3 so that no users associated with the identifier SALES have access to that terminal:

```
$ SET DEVICE/ACL=(IDENTIFIER=SALES,ACCESS=NONE) TTA3
```

If the device specified with the SET DEVICE/ACL command does not have an ACL, one is created.

The SET DEVICE/ACL command provides the following qualifiers to manipulate ACEs and ACLs:

/AFTER
/DELETE
/LIKE
/NEW
/REPLACE

You can delete ACEs from an ACL by including the /DELETE qualifier and specifying the ACEs with the /ACL qualifier. To delete all the ACEs (except those with the PROTECTED option), include the /DELETE qualifier and specify /ACL without specifying any ACEs.

You can copy an ACL from one object to another by using the /LIKE qualifier. The ACL of the object specified with /LIKE replaces the ACL of the device given with the command.

You can replace existing ACEs in the ACL of the device specified with the command by using the /REPLACE qualifier. Any ACEs specified with /ACL are deleted and replaced by those specified with /REPLACE.

The /NEW qualifier is used to delete all ACEs (except those with the PROTECTED option) before adding any ACEs specified by /ACL, /LIKE, or /REPLACE.

When referring to existing ACEs with /DELETE, /REPLACE, or /AFTER, the existing ACE may be abbreviated.

By default, any ACEs (except security alarm ACEs) added to an ACL are placed at the top of the ACL. Security alarm ACEs are always positioned at the top of the ACL, regardless of positioning qualifiers. Whenever the system receives a request for access to a device that has an ACL, the system searches each entry in the ACL from the first to the last for the first match it can find, and then stops searching. If another match occurs further down in the ACL, it will have no effect. Since the position of an ACE in an ACL is so important, you can use the /AFTER qualifier to correctly position an ACE. When you use the /AFTER qualifier, any ACEs added will be added after the ACE specified with /AFTER.

The SET DEVICE/ACL command can also be used with the /EDIT qualifier to invoke the ACL editor. The following qualifiers can be used only when the /EDIT qualifier has been specified.

/JOURNAL
/KEEP
/MODE
/RECOVER

QUALIFIERS

/AFTER=ace

Causes all access control entries (ACEs) specified with the /ACL qualifier to be added after the ACE specified with the /AFTER qualifier. By default, any ACEs added to the ACL are always placed at the top of the list.

This qualifier cannot be used with the /EDIT qualifier.

/DELETE

Indicates that the access control entries (ACEs) specified with the /ACL qualifier are to be deleted. If no ACEs are specified with /ACL, the entire ACL is deleted (except for ACEs with the PROTECTED option). If you specify an ACE that does not exist, you will be notified that the ACE does not exist, and the delete operation will continue.

This qualifier cannot be used with the /EDIT qualifier.

/EDIT

Invokes the ACL Editor and allows you to use the /JOURNAL, /KEEP, /MODE, or /RECOVER qualifiers. Any other qualifiers specified with /EDIT are ignored.

/JOURNAL[=file-spec]
/NOJOURNAL

Controls whether a journal file is created from the editing session. By default, a journal file is created if the editing session ends abnormally.

If you omit the file specification, the journal file has the same name as the input file and a file type of JOU. You can use the /JOURNAL qualifier to specify a journal file name that is different from the default. No wildcard characters are allowed in the /JOURNAL file-spec parameter.

You must specify /EDIT in order to use this qualifier.

/KEEP=(option[,...])

Determines whether the journal file or the recovery file will be deleted when the editing session ends. The options are:

- JOURNAL—saves the journal file for current editing session
- RECOVER—saves the journal file used for restoring the ACL

You can shorten the keywords JOURNAL and RECOVER to J and R, respectively. If you specify only one option, you can omit the parentheses.

You must specify /EDIT in order to use this qualifier.

/LIKE=object-spec

Indicates that the ACL of the object given with the /LIKE qualifier is to replace the ACL of the device specified with SET DEVICE/ACL. Any existing ACE (except those with the PROTECTED option) will be deleted before the ACL specified by /LIKE is copied.

No wildcard characters are allowed in the /LIKE device-name parameter.

This qualifier cannot be used with the /EDIT qualifier.

/LOG
/NOLOG (default)

Controls whether the SET DEVICE/ACL command displays the device name of the device that has been affected by the command.

This qualifier cannot be used with the /EDIT qualifier.

/MODE=[NO]PROMPT

Determines whether the ACL editor prompts for field values. By default, the ACL editor selects prompt mode.

You must specify the /EDIT qualifier to use this qualifier.

/NEW

Indicates that any existing ACE in the ACL of the device specified with SET DEVICE/ACL (except those with the PROTECTED option) is to be deleted. In order to use the /NEW qualifier, you must specify a new ACL or ACE with the /ACL, /LIKE, or /REPLACE qualifier.

This qualifier cannot be used with the /EDIT qualifier.

/RECOVER[=file-spec]
/NORECOVER (default)

Specifies the name of the journal file to be used in a recovery operation. If the file specification is omitted with /RECOVER, the journal file is assumed to have the same name as the input file and a file type of JOU. No wildcard characters are allowed with the /RECOVER file-spec parameter.

You must specify /EDIT in order to use this qualifier.

/REPLACE=(ace[,...])

Deletes the access control entries (ACEs) specified with the /ACL qualifier and replaces them with those specified with /REPLACE. Any ACEs specified with the /ACL qualifier must exist and must be specified in the order in which they appear in the ACL.

This qualifier cannot be used with the /EDIT qualifier.

EXAMPLES

1
```
$ SET DEVICE/ACL/LIKE=ABCD$ WRKD$
```

This example replaces the ACL of WRKD$ with the ACL for the device ABCD$.

2
```
$ SET DEVICE/ACL/EDIT/JOURNAL=ACL.JOB WORK3:
```

This SET DEVICE/ACL command invokes the interactive ACL editor and creates a journal file, named ACL.JOB, if the editing session ends abnormally. The /EDIT qualifier enables use of the /JOURNAL qualifier.

SET DIRECTORY/ACL

Allows you to modify the access control list (ACL) of one or more directories. The /ACL qualifier is required.

In VMS Version 5.0 the SET DIRECTORY/ACL command was replaced by SET ACL/OBJECT_TYPE=FILE. For more information on the replacement command see the Version 5.0 *VMS DCL Dictionary*.

FORMAT

SET DIRECTORY/ACL*[=(ace[,...])] directory-spec[,...]*

RESTRICTIONS

None.

PARAMETERS

directory-spec[,...]

Specifies one or more directories whose access control list (ACL) is being modified. Separate multiple directory specifications with commas. Device name and colon are optional. Wildcard characters are allowed in the directory specifications. Each directory must be a disk directory on a Files–11 Structure Level 2 formatted volume.

When the /EDIT qualifier is used, only one directory specification can be given, and it cannot include any wildcard characters.

(ace[,...])

Specifies one or more access control entries (ACEs) to be modified. When no ACE is specified, the entire access control list is affected. Separate multiple ACEs with commas. The specified ACEs are inserted at the top of the ACL unless the /AFTER qualifier is specified.

DESCRIPTION

The SET DIRECTORY/ACL command enables you to manipulate an entire access control list (ACL) of one or more directories, or to create, modify, or delete access control entries (ACEs) in the ACL of one or more directories. To use the SET DIRECTORY/ACL command, you specify the directory specification of the directory whose ACL you want to manipulate.

By using wildcard characters in the directory specification to the command, you can manipulate the ACLs of multiple directories with a single command. The following qualifiers can be used with wildcard characters to select a subset of the specified directories:

/BEFORE
/BY_OWNER
/CREATED
/EXCLUDE
/SINCE

You can also use the /CONFIRM qualifier to verify the directory selection.

The SET DIRECTORY/ACL command can be used to add ACEs to an ACL. For example, the following command adds an ACE to the ACL of the directory [.CONFIDENTIAL] so that all users associated with the identifier PERSONNEL are allowed read access to that directory:

```
$ SET DIRECTORY/ACL=(IDENTIFIER=PERSONNEL,ACCESS=READ) [.CONFIDENTIAL]
```

If the directory specified with the SET DIRECTORY/ACL command does not have an ACL, one is created.

The SET DIRECTORY/ACL command provides the following qualifiers to manipulate ACEs and ACLs:

/AFTER
/DELETE
/LIKE
/NEW
/REPLACE

You can delete ACEs from an ACL by including the /DELETE qualifier and specifying the ACEs with the /ACL qualifier. To delete all the ACEs (except those with the PROTECTED option), include the /DELETE qualifier and specify /ACL without specifying any ACEs.

You can copy an ACL from one object to a directory by using the /LIKE qualifier. The ACL of the object specified with /LIKE replaces the ACL of the directory given with the command.

You can replace existing ACEs in the ACL of the directory specified with the command by using the /REPLACE qualifier. Any ACEs specified with /ACL are deleted and replaced by those specified with /REPLACE.

The /NEW qualifier is used to delete all ACEs (except those with the PROTECTED option) before adding any ACEs specified by /ACL, /LIKE, or /REPLACE.

When referring to existing ACEs with /DELETE, /REPLACE, or /AFTER, the existing ACE may be abbreviated.

By default, any ACEs (except security alarm ACEs) added to an ACL are placed at the top of the ACL. Security alarm ACEs are always positioned at the top of the ACL, regardless of positioning qualifiers. Whenever the system receives a request for access to a directory that has an ACL, the system searches each entry in the ACL from the first to the last for the first match it can find and then stops searching. If another match occurs further down in the ACL, it will have no effect. Since the position of an ACE in an ACL is so important, you can use the /AFTER qualifier to correctly position an ACE. When you use the /AFTER qualifier, any ACEs added will be added after the ACE specified with /AFTER.

The SET DIRECTORY/ACL command can also be used with the /EDIT qualifier to invoke the ACL editor. When the /EDIT qualifier is specified, only one directory specification is allowed. When specifying the directory with /EDIT, use the syntax directory.DIR. The following qualifiers can be used only when the /EDIT qualifier has been specified.

/JOURNAL
/KEEP
/MODE
/RECOVER

QUALIFIERS

/AFTER=ace

Causes all access control entries (ACEs) specified with the /ACL qualifier to be added after the ACE specified with the /AFTER qualifier. By default any ACEs added to the ACL are always placed at the top of the list.

This qualifier cannot be used with the /EDIT qualifier.

/BEFORE[=time]

Selects only those directories that are dated before the specified time. You can specify either an absolute time or a combination of absolute and delta times. You can also use the keywords TODAY, TOMORROW, and YESTERDAY. If no time is specified, TODAY is assumed.

This qualifier cannot be used with the /EDIT qualifier.

/BY_OWNER[=uic]

Selects one or more directories whose owner user identification code (UIC) matches the specified owner UIC. If the /BY_OWNER qualifier is specified without a UIC, the UIC of the current process is assumed.

This qualifier cannot be used with the /EDIT qualifier.

/CONFIRM
/NOCONFIRM (default)

Controls whether a request is issued before each individual SET DIRECTORY /ACL operation to confirm that the operation should be performed on that directory.

When the system issues the prompt, you can issue any of the following responses:

YES	NO	QUIT
TRUE	FALSE	CTRL/Z
1	0	ALL
	<RET>	

You can use any combination of upper- and lowercase letters for word responses. Word responses can be abbreviated to one or more letters (for example, T, TR, or TRU for TRUE). Affirmative answers are YES, TRUE, and 1. Negative answers are NO, FALSE, 0, and <RET> . QUIT or CTRL/Z indicates that you want to stop processing the command at that point. When you respond with ALL, the command continues to process, but no further prompts are given. If you type a response other than one of those in the list, the prompt will be reissued.

This qualifier cannot be used with the /EDIT qualifier.

/CREATED

Selects directories based on their dates of creation. This qualifier is relevant only when used with the /BEFORE or /SINCE qualifier.

This qualifier cannot be used with the /EDIT qualifier.

/DEFAULT

Creates an ACL for the specified files as if the files were newly created. For a directory file, the /DEFAULT qualifier propagates the entire ACL (except ACEs with the NOPROPAGATE option) so that a particular access

protection can be propagated throughout a directory tree. For all other files, the /DEFAULT qualifier propagates the DEFAULT option ACEs in the ACL of the parent directory to the ACL of the specified files.

The /DEFAULT qualifier uses the ACL of the parent directory of the specified file, not the current default directory. This qualifier cannot be used with the /EDIT qualifier.

/DELETE

Indicates that the access control entries (ACEs) specified with the /ACL qualifier are to be deleted. If no ACEs are specified with /ACL, the entire ACL is deleted (except for ACEs with the PROTECTED option). If you specify an ACE that does not exist, you will be notified that the ACE does not exist and the delete operation will continue.

This qualifier cannot be used with the /EDIT qualifier.

/EDIT

Invokes the ACL Editor and allows you to use the /JOURNAL, /KEEP, /MODE, or /RECOVER qualifiers. Any other qualifiers specified with /EDIT are ignored. You can only supply one directory file specification with SET DIRECTORY/ACL/EDIT.

/EXCLUDE=(directory-spec[,...])

Excludes any directories that match the listed directory specifications from the SET DIRECTORY/ACL operation. If you specify only one directory, you can omit the parentheses. Wildcard characters are allowed in the directory specifications. The directory specification cannot contain a device name.

This qualifier cannot be used with the /EDIT qualifier.

/JOURNAL[=file-spec]
/NOJOURNAL

Controls whether a journal file is created from the editing session. By default, a journal file is created if the editing session ends abnormally.

If you omit the file specification, the journal file has the same name as the input file and a file type of JOU. You can use the /JOURNAL qualifier to specify a journal file name that is different from the default. No wildcard characters are allowed in the /JOURNAL file-spec parameter.

You must specify /EDIT in order to use this qualifier.

/KEEP=(option[,...])

Determines whether the journal file or the recovery file will be deleted when the editing session ends. The options are:

- JOURNAL—saves the journal file for the current editing session
- RECOVER—saves the journal file used for restoring the ACL

You can shorten the keywords JOURNAL and RECOVER to J and R, respectively. If you specify only one option, you can omit the parentheses.

You must specify /EDIT in order to use this qualifier.

/LIKE=object-spec

Indicates that the ACL of the object given with the /LIKE qualifier is to replace the ACL of the directories specified with SET DIRECTORY/ACL. Any existing ACE (except those with the PROTECTED option) will be deleted before the ACL specified by /LIKE is copied.

No wildcard characters are allowed in the /LIKE object-spec parameter.

This qualifier cannot be used with the /EDIT qualifier.

/LOG
/NOLOG (default)

Controls whether the SET DIRECTORY/ACL command displays the directory specification of each directory that has been affected by the command. By default, no log information is displayed.

This qualifier cannot be used with the /EDIT qualifier.

/MODE=[NO]PROMPT

Determines whether the ACL editor prompts for field values. By default, the ACL editor selects prompt mode.

You must specify the /EDIT qualifier to use this qualifier.

/NEW

Indicates that any existing ACE in the ACL of the directory specified with SET DIRECTORY/ACL (except those with the PROTECTED option) is to be deleted. In order to use the /NEW qualifier, you must specify a new ACL or ACE with the /ACL, /LIKE, or /REPLACE qualifier.

This qualifier cannot be used with the /EDIT qualifier.

/RECOVER[=file-spec]
/NORECOVER (default)

Specifies the name of the journal file to be used in a recovery operation. If the file specification is omitted with /RECOVER, the journal file is assumed to have the same name as the input file and a file type of JOU. No wildcard characters are allowed with the /RECOVER file-spec parameter.

You must specify /EDIT in order to use this qualifier.

/REPLACE=(ace[,...])

Deletes the access control entries (ACEs) specified with the /ACL qualifier and replaces them with those specified with /REPLACE. Any ACEs specified with the /ACL qualifier must exist and must be specified in the order in which they appear in the ACL.

This qualifier cannot be used with the /EDIT qualifier.

/SINCE[=time]

Selects only those directories that are dated after the specified time. You can specify either an absolute time or a combination of absolute and delta times. You can also use the keywords TODAY, TOMORROW, and YESTERDAY. If no time is specified, TODAY is assumed.

This qualifier cannot be used with the /EDIT qualifier.

EXAMPLES

1
```
$ SET DIRECTORY/ACL/LIKE=[.USER] [.USER.CAPTIVE_ACCOUNTS]
```

This example replaces the ACL of the directory CAPTIVE_ACCOUNTS with the ACL for the directory USER.LIS.

2
```
$ SET DIRECTORY/ACL=(IDENTIFIER=[123,321]+NETWORK,ACCESS=NONE) [.CONFIDENTIAL]
```

This command adds an ACE that specifies that NETWORK access for user [123,321] is not allowed for directory CONFIDENTIAL.

3
```
$ SET DIRECTORY/ACL/EDIT/JOURNAL=ACL.JOB [.PAYROLL]
```

This SET DIRECTORY/ACL command invokes the interactive ACL editor and creates a journal file, named ACL.JOB, if the editing session ends abnormally. The /EDIT qualifier enables use of the /JOURNAL qualifier.

4
```
$ SET DIRECTORY/ACL/EDIT/RECOVER=ACL.JOB [.PAYROLL]
```

This command uses the /RECOVER qualifier to restore the editing session from the previous example after the session ended abnormally.

SET FILE/ACL

Allows you to modify the access control list (ACL) of one or more files. The /ACL qualifier is required.

In VMS Version 5.0, the SET FILE/ACL command was replaced by SET ACL/OBJECT_TYPE=FILE. For more information on the replacement command see the Version 5.0 *VMS DCL Dictionary*.

FORMAT

SET FILE/ACL*[=(ace[,...])] file-spec[,...]*

RESTRICTIONS

None.

PARAMETERS

file-spec[,...]

Specifies one or more files whose access control list (ACL) is being modified. Separate multiple file specifications with commas. Wildcard characters are allowed in the file specifications. Each file must be a disk file on a Files-11 Structure Level 2 formatted volume.

When the /EDIT qualifier is used, only one file specification can be given, and it cannot include any wildcard characters.

(ace[,...])

Specifies one or more access control entries (ACEs) to be modified. When no ACE is specified, the entire access control list is affected. Separate multiple ACEs with commas. The specified ACEs are inserted at the top of the ACL unless the /AFTER qualifier is specified.

DESCRIPTION

The SET FILE/ACL command enables you to manipulate an entire access control list (ACL) of one or more files, or to create, modify, or delete access control entries (ACEs) in the ACL of one or more files. To use the SET FILE /ACL command, you specify the file specification of the file whose ACL you want to manipulate.

By using wildcard characters in the file specification to the command, you can manipulate the ACLs of multiple files with a single command. The following qualifiers can be used with wildcard characters to select a subset of the specified files:

/BEFORE
/BY_OWNER
/CREATED
/EXCLUDE
/SINCE

You can also use the /CONFIRM qualifier to verify the file selection.

The SET FILE/ACL command is used to add ACEs to an ACL. For example, the following command adds an ACE to the ACL of the file SALARY85.DAT so that all users associated with the identifier PERSONNEL are allowed read access to the file:

```
$ SET FILE/ACL=(IDENTIFIER=PERSONNEL,ACCESS=READ) SALARY85.DAT
```

If the file specified with the SET FILE/ACL command does not have an ACL, one is created.

The SET FILE/ACL command provides the following qualifiers to manipulate ACEs and ACLs:

/AFTER
/DELETE
/LIKE
/NEW
/REPLACE

You can delete ACEs from an ACL by including the /DELETE qualifier and specifying the ACEs with /ACL. To delete all the ACEs (except those with the PROTECTED option), include the /DELETE qualifier and specify /ACL without specifying any ACEs.

You can copy an ACL from one file to another by using the /LIKE qualifier. The ACL of the file specified with /LIKE replaces the ACL of the file given with the command.

You can replace existing ACEs in the ACL of the file specified with the command by using the /REPLACE qualifier. Any ACEs specified with /ACL are deleted and replaced by those specified with /REPLACE.

The /NEW qualifier is used to delete all ACEs (except those with the PROTECTED option) before adding any ACEs specified by /ACL, /LIKE, or /REPLACE.

When referring to existing ACEs with /DELETE, /REPLACE, or/AFTER, the existing ACE may be abbreviated.

By default, any ACEs (except security alarm ACE), added to an ACL are placed at the top of the ACL. Security alarm ACEs are always positioned at the top of the ACL, regardless of positioning qualifiers. Whenever the system receives a request for access to a file that has an ACL, the system searches each entry in the ACL from the first to the last for the first match it can find and then stops searching. If another match occurs further down in the ACL, it will have no effect. Since the position of an ACE in an ACL is so important, you can use the /AFTER qualifier to correctly position an ACE. When you use the /AFTER qualifier, any ACEs added will be added after the ACE specified with /AFTER.

The SET FILE command can also be used with the /EDIT qualifier to invoke the ACL editor. When the /EDIT qualifier is specified, only one file specification is allowed. The following qualifiers can be used only when the /EDIT qualifier has been specified.

/JOURNAL
/KEEP
/MODE
/RECOVER

QUALIFIERS

/AFTER=ace

Causes all access control entries (ACEs) specified with the /ACL qualifier to be added after the ACE specified with the /AFTER qualifier. By default, any ACEs added to the ACL are always placed at the top of the list.

This qualifier cannot be used with the /EDIT qualifier.

/BEFORE[=time]

Selects only those files that are dated before the specified time. You can specify either an absolute time or a combination of absolute and delta times. You can also use the keywords TODAY, TOMORROW, and YESTERDAY. If no time is specified, TODAY is assumed.

This qualifier cannot be used with the /EDIT qualifier.

/BY_OWNER[=uic]

Selects one or more files whose owner user identification code (UIC) matches the specified owner UIC. If the /BY_OWNER qualifier is specified without a UIC, the UIC of the current process is assumed.

This qualifier cannot be used with the /EDIT qualifier.

/CONFIRM
/NOCONFIRM (default)

Controls whether a request is issued before each individual SET FILE/ACL operation to confirm that the operation should be performed on that file.

When the system issues the prompt, you can issue any of the following responses:

YES	NO	QUIT
TRUE	FALSE	CTRL/Z
1	0	ALL
	<RET>	

You can use any combination of upper- and lowercase letters for word responses. Word responses can be abbreviated to one or more letters (for example, T, TR, or TRU for TRUE). Affirmative answers are YES, TRUE, and 1. Negative answers are NO, FALSE, 0, and <RET> . QUIT or CTRL/Z indicates that you want to stop processing the command at that point. When you respond with ALL, the command continues to process, but no further prompts are given. If you type a response other than one of those in the list, the prompt will be reissued.

This qualifier cannot be used with the /EDIT qualifier.

/CREATED

Selects files based on their dates of creation. This qualifier is relevant only when used with the /BEFORE or /SINCE qualifier.

This qualifier cannot be used with the /EDIT qualifier.

/DEFAULT

Creates an ACL for the specified files as if the files were newly created. For a directory file, the /DEFAULT qualifier propagates the entire ACL (except ACEs with the NOPROPAGATE option) so that a particular access protection can be propagated throughout a directory tree. For all other files,

the /DEFAULT qualifier propagates the DEFAULT option ACEs in the ACL of the parent directory to the ACL of the specified files.

The /DEFAULT qualifier uses the ACL of the parent directory of the specified file, not the current default directory. This qualifier cannot be used with the /EDIT qualifier.

/DELETE

Indicates that the access control entries (ACEs) specified with the /ACL qualifier are to be deleted. If no ACEs are specified with /ACL, the entire ACL is deleted (except for ACEs with the PROTECTED option). If you specify an ACE that does not exist with the /ACL qualifier, you will be notified that the ACE does not exist and the delete operation will continue.

This qualifier cannot be used with the /EDIT qualifier.

/EDIT

Invokes the ACL Editor and allows you to use the /JOURNAL, /KEEP, /MODE, or /RECOVER qualifiers. Any other qualifiers specified with /EDIT are ignored. You can only supply one file specification with SET FILE/ACL /EDIT.

/EXCLUDE=(file-spec[,...])

Excludes any files that match the listed file specifications from the SET FILE /ACL operation. If you specify only one file, you can omit the parentheses. Wildcard characters are allowed in the file specifications. However, you cannot use relative version numbers to exclude a specific version. The file specification can contain a directory specification; however, the file specification cannot contain a device name.

This qualifier cannot be used with the /EDIT qualifier.

/JOURNAL[=file-spec]
/NOJOURNAL

Controls whether a journal file is created from the editing session. By default, a journal file is created if the editing session ends abnormally.

If you omit the file specification, the journal file has the same name as the input file and a file type of JOU. You can use the /JOURNAL qualifier to specify a journal file name that is different from the default. No wildcard characters are allowed in the /JOURNAL file-spec parameter.

You must specify /EDIT in order to use this qualifier.

/KEEP=(option[,...])

Determines whether the journal file or the recovery file will be deleted when the editing session ends. The options are:

- JOURNAL—saves the journal file for current editing session
- RECOVER—saves the journal file used for restoring the ACL

You can shorten the keywords JOURNAL and RECOVER to J and R, respectively. If you specify only one option, you can omit the parentheses.

You must specify /EDIT in order to use this qualifier.

/LIKE=file-spec

Indicates that the ACL of the file given with the /LIKE qualifier is to replace the ACL of the files specified with SET FILE/ACL. Any existing ACE (except those with the PROTECTED option) will be deleted before the ACL specified by /LIKE is copied.

No wildcard characters are allowed in the /LIKE file-spec parameter.

This qualifier cannot be used with the /EDIT qualifier.

/LOG
/NOLOG (default)

Controls whether the SET FILE/ACL command displays the file specification of each file that has been affected by the command.

This qualifier cannot be used with the /EDIT qualifier.

/MODE=[NO]PROMPT

Determines whether the ACL editor prompts for field values. By default, the ACL editor selects prompt mode.

You must specify the /EDIT qualifier to use this qualifier.

/NEW

Indicates that any existing ACE in the ACL of a file specified with SET FILE /ACL (except those with the PROTECTED option) is to be deleted. In order to use the /NEW qualifier, you must specify a new ACL or ACE with the /ACL, /LIKE, or /REPLACE qualifier.

This qualifier cannot be used with the /EDIT qualifier.

/RECOVER[=file-spec]
/NORECOVER (default)

Specifies the name of the journal file to be used in a recovery operation. If the file specification is omitted with /RECOVER, the journal file is assumed to have the same name as the input file and a file type of JOU. No wildcard characters are allowed with the /RECOVER file-spec parameter.

You must specify /EDIT in order to use this qualifier.

/REPLACE=(ace[,...])

Deletes the access control entries (ACEs) specified with the /ACL qualifier and replaces them with those specified with /REPLACE. Any ACEs specified with the /ACL qualifier must exist and must be specified in the order in which they appear in the ACL.

This qualifier cannot be used with the /EDIT qualifier.

/SINCE[=time]

Selects only those files that are dated after the specified time. You can specify either an absolute time or a combination of absolute and delta times. You can also use the keywords TODAY, TOMORROW, and YESTERDAY. If no time is specified, TODAY is assumed.

This qualifier cannot be used with the /EDIT qualifier.

EXAMPLES

1
```
$ SET FILE/ACL/LIKE=USER.LIS CAPTIVE_ACCOUNTS.LIS
```

This example replaces the ACL of the file CAPTIVE_ACCOUNTS.LIS with the ACL for the file USER.LIS.

2
```
$ SET FILE/ACL=(IDENTIFIER=[123,321]+NETWORK,ACCESS=NONE) *.*
```

This command adds an ACE that specifies that NETWORK access for user [123,321] is not allowed for each file in the default directory.

3
```
$ SET FILE/ACL=(IDENTIFIER=[SALES,FRANK],ACCESS=READ)/DELETE *.*
```

This SET FILE/ACL command deletes the specified ACE from all files in the default directory.

4
```
$ SET FILE/ACL/EDIT/JOURNAL=ACL.JOB PASSWORD_2.DAT
```

This SET FILE/ACL command invokes the interactive ACL editor and creates a journal file, ACL.JOB, if the editing session ends abnormally. The /EDIT qualifier enables use of the /JOURNAL qualifier.

5
```
$ SET FILE/ACL/EDIT/RECOVER=ACL.JOB PASSWORD_2.DAT
```

This command uses the /RECOVER qualifier to restore the editing session from the previous example after the session ended abnormally.

SET QUEUE/ENTRY

Changes the current status or attributes of a job that is not currently executing in a queue. The /ENTRY qualifier is required.

In VMS Version 5.0, the SET QUEUE/ENTRY command was replaced by SET ENTRY. For more information on the replacement command see Version 5.0 *VMS DCL Dictionary*.

FORMAT

SET QUEUE/ENTRY=*entry-number queue-name[:]*

RESTRICTIONS

Requires operator (OPER) privilege or execute (E) access to the specified queue. If you have D access to the specified job, you can alter the attributes for that job.

PARAMETERS

entry-number

Specifies the entry number of the job you want to change.

queue-name[:]

Specifies the name of the queue in which the specified job is entered.

DESCRIPTION

The SET QUEUE/ENTRY command allows you to change the status or attributes of a job that has been submitted to a printer or batch queue, as long as the job is not currently executing. (You cannot affect individual files within a multifile job with the SET/QUEUE/ENTRY command.)

The qualifiers enable you to specify different attributes or delete attributes. Some qualifiers apply to both batch and print jobs. Others are restricted to either batch jobs or print jobs. The defaults for all the SET QUEUE/ENTRY qualifiers are the attributes and status that the job has before you issue the SET QUEUE/ENTRY command.

The system assigns a unique entry number to each queued print or batch job in the system. The PRINT and SUBMIT commands display the job number when they successfully queue a job for processing. You can issue the SHOW QUEUE command to refresh your memory about a job's entry number. Use the job entry number to specify which job you want to change.

QUALIFIERS

/AFTER=time
/NOAFTER

Requests that the specified job be held until after a specific time. If the specified time has already passed, the job is queued for immediate processing.

You can specify either an absolute time or a combination of absolute and delta times.

/BURST[=keyword]
/NOBURST

Controls whether a burst page is included at the beginning of a print job. A burst page precedes a flag page and contains the same information. However, it is printed over the perforation between the burst page and the flag page. The printing on the perforation makes it easy to separate individual print jobs.

When you specify /BURST, you need not specify /FLAG; a flag page will automatically follow the burst page.

You can specify one of the following keywords:

ALL	All printed files contain a burst page.
ONE	The first printed file contains a burst page.

Use the /[NO]BURST qualifier to override the installation-defined defaults that have been set for the printer queue you are using.

/CHARACTERISTICS=(characteristic[,...])
/NOCHARACTERISTICS

Enables you to change the characteristics desired for the job. If you specify only one characteristic, you can omit the parentheses. Codes for characteristics can be either names or values from 0 to 127 and are installation-defined. Use the SHOW QUEUE/CHARACTERISTICS command to see which characteristics have been defined for your system. Use the SHOW QUEUE/FULL command to see which characteristics are available on a particular queue.

When you include the /CHARACTERISTICS qualifier with the SET QUEUE /ENTRY command, all the characteristics you specify must also be specified for the queue that will be executing the job. If not, the job will remain pending in the queue until the queue characteristics are changed or you delete the entry with the DELETE/ENTRY command. You need not specify every characteristic of a queue with the SET QUEUE/ENTRY command as long as the ones you specify are a subset of the characteristics set for that queue. The job will also run if no characteristics are specified.

Specification of a characteristic for a queue does not prevent jobs that do not specify that characteristic from being executed.

/CLI=filename

Enables you to specify a different command language interpreter (CLI) to use in processing the job. The file name specifies that the CLI be SYS$SYSTEM:filename.EXE. If you do not specify the /CLI qualifier, the job is run by the CLI specified in the user's authorization record, or whatever CLI was specified when the job was originally submitted to the queue.

/COPIES=n

Specifies the number of copies to print. The n parameter can be any number from 1 to 255.

When you use the /COPIES qualifier with the SET QUEUE/ENTRY command, the number of copies can apply only to the entire job. You cannot use this qualifier to specify different numbers of copies for individual files within a multifile job.

/CPUTIME=option

Defines a CPU time limit for the batch job. You can specify a delta time, the value 0, or the keyword NONE or INFINITE for n.

When you need less CPU time than authorized, use the /CPUTIME qualifier to override the base queue value established by the system manager or the value authorized in your user authorization file. Specify 0 or INFINITE to request an infinite amount of time. Specify NONE when you want the CPU time to default to your user authorization file (UAF) value or the limit specified on the queue. Note that you cannot request more time than permitted by the base queue limits or your own UAF.

/FEED
/NOFEED

Controls whether form feeds are inserted into print jobs when the printer nears the end of a page. The number of lines per form can be reset by the /FORM qualifier. You can suppress this automatic form feed (without affecting any of the other carriage control functions that are in place) by using the /NOFEED qualifier.

When you use the /FEED qualifier with the SET QUEUE/ENTRY command, the qualifier applies to all files in the print job. You cannot use this qualifier to specify form feeds for individual files within a multi-file job.

/FLAG[=keyword]
/NOFLAG

Controls whether a flag page is printed preceding a print job. The flag page contains the name of the user submitting the job, the job entry number, and other information about the job. You can specify one of the following keywords:

ALL	Prints a flag page before each file in the job
ONE	Prints a flag page before the first file in the job

Use the /[NO]FLAG qualifier to override the installation-defined defaults that have been set for the printer queue you are using.

/FORM=type

Specifies the name of the form that you want for the print job.

Specify the form type using a numeric value or alphanumeric code. Form types can refer to the width, length, or type of paper. Codes for form types are installation-defined. You can use the SHOW QUEUE/FORM command to find out the form types available for your system. The SHOW QUEUE/FULL command tells you which form is set for a specific queue.

If you specify a form type different from that of the queue, your job remains pending until the form type of the queue is set equal to the form type of the job or you delete the job with the DELETE/ENTRY command. You can use the SET QUEUE/ENTRY to change the form type of your job to match that of the queue so your job can be printed. In order to have the form type for the queue changed, request that the system manager stop the queue, physically change the form type of the printer, and restart the queue specifying the new form type.

/HEADER
/NOHEADER

Controls whether a heading line is printed at the top of each output page in a print job.

/HOLD
/NOHOLD

Controls whether or not the job is to be made available for immediate processing or held for processing later.

If you specify /HOLD, the job is not released for processing until you specifically release it with the /NOHOLD or /RELEASE qualifier. You can use the SET QUEUE/ENTRY command to release a job that was previously submitted with a /HOLD qualifier or you can place a job on hold so that it will run later.

You can use the /NOHOLD qualifier to release jobs that have been held for the following reasons:

- A job was submitted with the /HOLD qualifier.
- A completed job is being held in a queue that has /RETAIN specified.
- A user-written symbiont has refused a job.

/JOB_COUNT=n

Requests that an entire print job be printed n times, where n is a decimal integer from 1 to 255. This qualifier overrides the /JOB_COUNT qualifier specified or defaulted with the PRINT command.

/KEEP
/NOKEEP

Controls whether the batch job log file is deleted after it is printed.

/LOG_FILE=file-spec
/NOLOG_FILE

Controls whether a log file with the specified name is created for the batch job or whether a log file is created.

When you use the /LOG_FILE qualifier, the system writes the log file to the file you specify. If you use /NOLOG_FILE, no log file is created. If neither form of the qualifier has been used for the job, the log file is written to a file in the default directory that has the same file name as the first command file and a file type of LOG.

You can use the /LOG_FILE qualifier to specify that the log file be written to a different device. Logical names in the file specification are translated in the context of the process that executes the SET QUEUE/ENTRY command. The process executing the batch job must have access to the device on which the log file will reside.

If you omit the /LOG_FILE qualifier and specify the /NAME qualifier, the log file is written to a file having the same file name as that specified by the /NAME qualifier; the file type is LOG. When you omit the /LOG_FILE qualifier, the job-name value used with /NAME must be a valid file name.

/LOWERCASE
/NOLOWERCASE

Indicates whether the files must be printed on a printer that can print both uppercase and lowercase letters. The /NOLOWERCASE qualifier means that files can be printed on printers supporting only uppercase letters. If all available printers can print both uppercase and lowercase letters, you do not need to specify /LOWERCASE.

/NAME=job-name

Defines a name string to identify the job. The name string can have from 1 to 39 characters. The job name is used in the SHOW QUEUE command display. For batch jobs, the job name is also used for the batch job log file. For print jobs, the job name is also used on the flag page of the printed output.

If the /NAME qualifier has not been specified for the job, the name string defaults to the file name of the first, or only, file in the job; the file type is LOG.

/NOCHECKPOINT

For a batch job, erases the value established by the most recently executed SET RESTART_VALUE command. For a print job, clears the stored checkpoint so that the job will restart from the beginning.

/NODELETE

Cancels file deletion for a job that was submitted with the /DELETE qualifier. If no /DELETE qualifier was specified when the job was originally submitted to the queue, you cannot use the SET QUEUE/ENTRY to establish file deletion at a later time.

You cannot use the /NODELETE qualifier to specify that individual files in a multi-file job not be deleted.

/NOTE=string

Allows you to specify a message to appear on the flag page for the print job. The string can contain up to 255 characters.

/NOTIFY
/NONOTIFY

Controls whether a message is broadcast to any terminal at which you are logged in, notifying you when your job has been completed or aborted.

/OPERATOR=string

Allows you to specify a message to be sent to the operator. The string can contain up to 255 characters.

When the job begins execution, the queue pauses and the message is transmitted to the operator.

/PAGES=([l,]u)

Specifies the number of pages to print for the specified job. You can use the /PAGES qualifier to print portions of a long file.

When you use the /PAGES qualifier with the SET QUEUE/ENTRY command, the qualifier can only apply to an entire job. You cannot use this qualifier to specify different numbers of pages to be printed for individual files within a multi-file job.

The l (lower) specifier refers to the first page in the group of pages that you want printed for that job. If you omit the l specifier, the printing starts on the first page of the job. The u (upper) specifier refers to the last page of the file that you want printed. When you want to print to the end of the file but do not know how many pages that will be, you can use "" as the u specifier. You can omit the parentheses when you specify only a value for u. For example, /PAGES=10 prints the first 10 pages of the job; /PAGES=(5,10) prints pages 5 through 10; /PAGES=(5,"") starts printing at page 5 and continues until the end of the job is reached.

/PARAMETERS=(parameter[,...])

Specifies from 1 to 8 optional parameters to be passed to the job. Each parameter can have as many as 255 characters.

If you specify only one parameter, you can omit the parentheses. The commas delimit individual parameters. To specify a parameter that contains any special characters or delimiters, enclose the parameter in quotation marks.

For batch jobs, the parameters define values to be equated to the symbols named P1 through P8 in each command procedure in the job. The symbols are local to the specified command procedures.

/PASSALL
/NOPASSALL

Specifies whether the symbiont bypasses all formatting and sends the output QIO to the driver with format suppressed. All qualifiers affecting formatting, as well as the /HEADER, /PAGES, and /PAGE_SETUP qualifiers, will be ignored.

When you use the /PASSALL qualifier with the SET QUEUE/ENTRY command, the qualifier applies to the entire job. You cannot use this qualifier to specify PASSALL mode for individual files within a multifile job.

/PRINTER[=queue-name]
/NOPRINTER

Controls whether the batch job log is queued for printing when your job is completed. The /PRINTER qualifier allows you to specify a particular printer queue.

If you specify /NOPRINTER, /KEEP is assumed.

/PRIORITY=n

Specifies the priority of the job. The priority value must be in the range of 0 through 255, where 0 is the lowest priority and 255 is the highest.

The default value for /PRIORITY is the value of the SYSGEN parameter DEFQUEPRI. You must have either OPER (operator) or ALTPRI (alter priority) privilege to raise the priority value above the value of the SYSGEN parameter MAXQUEPRI. No privilege is needed to set the priority lower than the MAXQUEPRI value.

/RELEASE

Releases a previously held job for processing. You can use this qualifier to release jobs that have been held for the following reasons:

- A job was submitted with the /HOLD qualifier.
- A job was submitted with the /AFTER qualifier.

- A completed job is being held in a queue that has /RETAIN specified.
- A user-written symbiont has refused a job.

/REQUEUE=queue-name[:]

Requests that the job be moved from the original queue to the specified queue.

/RESTART
/NORESTART

Specifies whether a batch or print job will be restarted after a system crash or a STOP/QUEUE/REQUEUE command.

/SETUP=module[,...]

Calls for the specified modules to be extracted from the device control library and copied to the printer before a job is printed.

When you use the /SETUP qualifier with the SET QUEUE/ENTRY command, the qualifier applies to the entire job. You cannot use this qualifier to specify different setup modules for individual files within a multi-file job.

/SPACE
/NOSPACE

Controls whether output is to be double-spaced.

When you use the /SPACE qualifier with the SET QUEUE/ENTRY command, the qualifier applies to the entire job. You cannot use this qualifier to specify different spacing for individual files within a multi-file job.

/TRAILER[=keyword]
/NOTRAILER

Controls whether a trailer page is printed at the end of a job. The trailer page displays the job entry number, as well as information about the user submitting the job.

When you use the /TRAILER qualifier with the SET QUEUE/ENTRY command, trailer pages are placed at the end of each file in a multi-file job. You can specify one of the following keywords:

ALL	All printed files contain a trailer page.
ONE	The last printed file contains a trailer page.

Use the /[NO]TRAILER qualifier to override the installation-defined defaults that have been set for the printer queue you are using.

/WSDEFAULT=n

Defines a working set default for a batch job. You can specify a positive integer in the range 1 through 65,535, 0, or the word NONE for n.

Use this qualifier to override the base queue value established by the system manager or the value authorized in the user authorization file (UAF), provided you want to impose a lower value. Specify 0 or NONE if you want the working set value defaulted to either the UAF value or the working set quota specified on the queue. You cannot request a value higher than the default.

/WSEXTENT=n

Defines a working set extent for a batch job. You can specify a positive integer in the range 1 through 65,535, 0, or the word NONE for n.

Use this qualifier to override the base queue value established by the system manager or the value authorized in the user authorization file (UAF), provided you want to impose a lower value. Specify 0 or NONE if you want the working set extent defaulted to either the UAF or the working set extent specified on the queue. You cannot request a value higher than the default.

/WSQUOTA=n

Defines the maximum working set size for a batch job. This is the working set quota. You can specify a positive integer in the range 1 through 65,535, 0, or the word NONE for n.

Use this qualifier to override the base queue value established by the system manager or the value authorized in the user authorization file (UAF), provided you want to impose a lower value. Specify 0 or NONE if you want the working set quota defaulted to either the user authorization file value or the working set quota specified on the queue. You cannot request a value higher than the default.

EXAMPLES

1

```
$ PRINT/HOLD   MYFILE.DAT
  Job MYFILE (queue SYS$PRINT, entry 112) holding
    .
    .
    .
$ SET QUEUE/ENTRY=112/RELEASE/JOB_COUNT=3
```

The PRINT command requests that the file MYFILE.DAT be queued to the system printer, but placed in a hold status. The SET QUEUE/ENTRY command releases the file for printing and changes the number of copies of the job to three.

2

```
$ SUBMIT  WEATHER
  Job WEATHER (queue SYS$BATCH, entry 210) pending
$ SUBMIT  CLIMATE
  Job CLIMATE (queue SYS$BATCH, entry 211) pending
$ SET  QUEUE/ENTRY=211/HOLD/NAME=TEMP  SYS$BATCH
```

The two SUBMIT commands queue command procedures for batch processing. The system assigns them job numbers of 210 and 211, respectively. The SET QUEUE/ENTRY command places the second job in a hold state and changes the job name to TEMP, assuming that job 211 had not yet begun execution.

3
```
$ PRINT/FLAG=ALL/AFTER=20:00 MEMO.MEM, LETTER.MEM, REPORT.MEM/SPACE
Job MEMO (queue SYS$PRINT, entry 172) holding until 20:00
.
.
.
$ SET QUEUE/ENTRY=172 /BURST/NOSPACE/HEADER SYS$PRINT
```

The PRINT command requests that three files be printed after 8:00 P.M. on the default printer with flag pages preceding each file. It also requests that the file REPORT.MEM be double-spaced. Later a SET QUEUE/ENTRY command is issued. This command calls for a burst page at the beginning of each file and requests that all files in the job be single-spaced. Headers are printed on each page of each file in the job.

1.2 Obsolete DCL Command Qualifiers

This section describes obsolete DCL command qualifiers for current DCL commands. Table 1–2 lists the obsolete command string in the left column, the version of VMS that superseded the obsolete command string in the center column, and the current command string in the third column. For example, INITIALIZE/QUEUE/FLAG was replaced in VMS Version 4.0 by INITIALIZE/QUEUE/SEPARATE=FLAG.

Table 1–2 Table of Obsolete DCL Command Qualifiers

Obsolete Command String	Made Obsolete in Version	Current Command String
INITIALIZE/QUEUE/FLAG	4.0	INITIALIZE/QUEUE/SEPARATE=FLAG
/BURST	4.0	/SEPARATE=BURST
/PRIORITY	4.0	/BASE_PRIORITY
/TERMINAL	5.0	/DEVICE=TERMINAL
START/QUEUE/BATCH	5.0	INITIALIZE/QUEUE/BATCH
/PRIORITY	4.0	START/QUEUE/BASE_PRIORITY
/TERMINAL	5.0	INITIALIZE/QUEUE/DEVICE=TERMINAL

1.2.1 Obsolete Qualifiers of the INITIALIZE/QUEUE Command

This section describes the following obsolete qualifiers of the INITIALIZE /QUEUE command:

- /[NO]BURST
- /[NO]FLAG
- /PRIORITY
- /TERMINAL

1.2.1.1 INITIALIZE/QUEUE/[NO]BURST

The /BURST qualifier of the INITIALIZE/QUEUE command has been replaced by the /SEPARATE=BURST qualifier. For more information on the replacement qualifier see the Version 5.0 *VMS DCL Dictionary*.

/BURST
/NOBURST

Controls whether a burst header page is printed for each print job.

Use the /BURST qualifier to print a header page over the paper perforations so that the page header is visible from the side of a stack of paper. A burst header uses an extra page for each job but simplifies separating listings.

The default is /NOBURST.

1.2.1.2 INITIALIZE/QUEUE/[NO]FLAG

The /[NO]FLAG qualifier of the INITIALIZE/QUEUE command has been replaced by the /SEPARATE=FLAG qualifier. For more information on the replacement qualifier see the Version 5.0 *VMS DCL Dictionary*.

/FLAG
/NOFLAG

Specifies whether a header page is printed at the beginning of the first file in each print job.

The default is /NOFLAG.

1.2.1.3 INITIALIZE/QUEUE/PRIORITY

The /PRIORITY qualifier of the INITIALIZE/QUEUE command has been replaced by the /BASE_PRIORITY qualifier. For more information on the replacement qualifier see the Version 5.0 *VMS DCL Dictionary*.

/PRIORITY=n

Specifies the base process priority at which jobs are initiated from a batch queue. By default, if you omit the qualifier, jobs are initiated at the base priority established by the DEFPRI system generation parameter.

1.2.1.4 INITIALIZE/QUEUE/TERMINAL

The /TERMINAL qualifier of the INITIALIZE/QUEUE command has been replaced by the /DEVICE=TERMINAL qualifier. For more information on the replacement qualifier see the Version 5.0 *VMS DCL Dictionary*.

/TERMINAL

Indicates that a generic queue will be associated with terminal queues instead of printer queues. The /TERMINAL qualifier allows all jobs entered in the generic queue to be moved to terminal queues with matching settings.

1.2.2 Obsolete Qualifiers of the START/QUEUE Command

This section describes the following obsolete qualifiers of the START/QUEUE command:

- /[NO]BATCH
- /PRIORITY
- /TERMINAL

1.2.2.1 START/QUEUE/[NO]BATCH

The /BATCH qualifier of the START/QUEUE command has been replaced by the INITIALIZE/QUEUE/BATCH command string. For more information on the replacement command string see the Version 5.0 *VMS DCL Dictionary*.

/BATCH
/NOBATCH (default)

Indicates that you are starting a batch queue. You cannot use the /BATCH qualifier unless the queue you are starting was initialized as a batch queue. If an existing queue is a batch queue, you can optionally use the /BATCH qualifier.

1.2.2.2 START/QUEUE/PRIORITY=n

The /PRIORITY qualifier of the START/QUEUE command has been replaced by the /BASE_PRIORITY qualifier. For more information on the replacement qualifier see the Version 5.0 *VMS DCL Dictionary*.

/PRIORITY=n

Specifies the base process priority at which jobs are initiated from a batch queue. By default, if you omit the qualifier, jobs are initiated at the base priority established by the DEFPRI system generation parameter.

1.2.2.3 START/QUEUE/TERMINAL

The /TERMINAL qualifier of the START/QUEUE command has been replaced by the INITIALIZE/QUEUE/DEVICE=TERMINAL command string. For more information on the replacement command string see the Version 5.0 *VMS DCL Dictionary*.

/TERMINAL
/NOTERMINAL

Indicates that a generic queue will be associated with terminal queues instead of printer queues. The /TERMINAL qualifier allows all jobs entered in the generic queue to be moved to terminal queues with matching settings.

The /NOTERMINAL qualifier cancels the effect of a previous /TERMINAL setting.

2 Obsolete System Services

This chapter describes obsolete system services. An *obsolete service* is one whose functions have been superseded by a new service. Obsolete services are no longer updated.

DIGITAL recommends that you use current services when you write new programs. Old programs that use obsolete services should be updated with current services.

The following table displays the obsolete services described in this chapter and the current services that have replaced them.

Table 2–1 Table of Obsolete System Services

Obsolete Service	Current Service
$BRDCST	$BRKTHRU, $BRKTHRUW
$CNTREG	$DELTVA
$CRELOG	$CRELNM
$DELLOG	$DELLNM
$GETCHN	$GETDVI, $GETDVIW
$GETDEV	$GETDVI, $GETDVIW
$INPUT	$QIO,$QIOW
$OUTPUT	$QIO,$QIOW
$SNDACC	$SNDJBC, $SNDJBCW
$SNDSMB	$SNDJBC, $SNDJBCW
$TRNLOG	$TRNLNM

$BRDCST Broadcast

The Broadcast service writes a message to one or more terminals.

The Breakthrough ($BRKTHRU) and Breakthrough and Wait ($BRKTHRUW) services supersede the $BRDCST service. New programs should be written using $BRKTHRU or $BRKTHRUW, not $BRDCST, and old programs using $BRDCST should be changed to use $BRKTHRU or $BRKTHRUW.

FORMAT

SYS$BRDCST *msgbuf [,devnam] [,flags] [,carcon]*

RETURNS

VMS usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

Longword condition value. All system services return (by value) a condition value in R0. Condition values that can be returned by this service are listed under "CONDITION VALUES RETURNED."

ARGUMENTS

msgbuf

VMS usage: **char_string**
type: **character-coded text string**
access: **read only**
mechanism: **by descriptor–fixed length string descriptor**

Message text to be sent to the specified terminal(s). The **msgbuf** argument is the address of a descriptor pointing to this message text.

The $BRDCST service permits the message text to be as long as 16,350 bytes; however, both the SYSGEN parameter MAXBUF and the caller's buffered I/O byte count limit (BYTLM) quota may affect the maximum length of the message text.

devnam

VMS usage: **device_name**
type: **character-coded text string**
access: **read only**
mechanism: **by descriptor–fixed length string descriptor**

Terminal name to which to send the message. The **devnam** argument is the address of a character string descriptor pointing to the terminal name.

If this argument is omitted or specified as 0, the message is broadcast to all terminals.

If the first longword in the descriptor contains a 0, the message is sent to all terminals that are currently allocated to processes.

flags

VMS usage: **mask_longword**
type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

Flag bit mask specifying (1) positioning options for the message and (2) the number of screen lines to clear before writing the message (only for terminals that have set the DEC_CRT characteristic). The **flags** argument is a longword value that is the logical OR of each desired flag option and of the number of lines to be cleared.

Terminals set the DEC_CRT characterististic by issuing the DCL command SET TERMINAL/DEC_CRT. If a terminal has not set the DEC_CRT characteristic, $BRDCST positions the message on the screen using the options specified in **flags**, but it does not clear any lines before writing the message.

If the **flags** argument is not specified, $BRDCST positions the message by using the information supplied in the **carcon** argument (if specified) or by using the default carriage control and line feed (if **carcon** is not specified).

Each flag option has a symbolic name. These symbolic names are defined by the $BRDCSTDEF macro. The following lists the symbolic name and description of each flag option.

Flag Option	Description
BRDCST$M_SCREEN	When specified, $BRDCST writes the message to the top of the terminal screen, issues a carriage control and line feed, redisplays the last line of a read operation (if one was interrupted by the broadcast message), and repositions the cursor to its position prior to the broadcast message.
BRDCST$M_ BOTTOM	When BRDCST$M_BOTTOM is specified and when BRDCST$M_SCREEN is also specified, $BRDCST writes the message to the bottom of the terminal screen. If BRDCST$M_SCREEN is not also specified, $BRDCST writes the message to the line where the cursor is currently positioned.
BRDCST$M_REFRSH	When specified, $BRDCST, after writing the message to the screen, does not redisplay the last line of a read operation that was interrupted by the broadcast message.

carcon

VMS usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

Carriage control specifier indicating the carriage control sequence that is to follow the message that $BRDCST sends to the terminal(s). The **carcon** argument is a longword containing the carriage control specifier.

If the **carcon** argument is not specified, $BRDCST follows the message with a carriage return and line feed.

The **carcon** argument should only be specified if the BRDCST$M_SCREEN flag in the **flags** argument is not set; since if this flag is set, $BRDCST automatically formats the screen. Refer to the description of this flag in the **flags** argument for more information.

DESCRIPTION

The calling process must have OPER privilege to send a message to either more than one terminal or to a terminal that is allocated to another user.

The $BRDCST service requires system dynamic memory; it also uses the process's buffered I/O byte count limit (BYTLM) quota to buffer the message while the service executes.

The service does not return control to the caller until all specified terminals have displayed the broadcast message.

The current state of the terminal determines if and when the broadcast message is displayed on the screen:

1. If the terminal is reading when $BRKTHRU sends the message, the read operation is suspended, the message is displayed, and then the line that was being read when the read operation was suspended is redisplayed (equivalent to the action produced by CTRL/R).
2. If the terminal is writing when $BRKTHRU sends the message, the message is displayed after the current write operation has completed.
3. The message is not displayed in any of the following cases:
   - The terminal has set the NOBROADCAST characteristic, for all images, or for the image that is currently requesting the broadcast.
   - The terminal is in PASSALL mode.
   - The terminal's current operation is *read physical block* (IO$_READPBLK function code).
   - The terminal's current operation has specified *no echo* (IO$_NOECHO function modifier).
   - The terminal's current operation has specified *no format* (IO$_NOFORMAT function modifier).

After the message is displayed, the terminal is returned to the state it was in prior to receiving the message.

Terminals with the DEC_CRT characteristic can be directed to clear a number of lines before the message is broadcast. To specify the number of lines, use the logical OR of the number of lines and the flags mask. For example, to clear four lines on a DEC_CRT terminal, the following value would be specified:

```
<#4!BRDCST$M_SCREEN>
```

CONDITION VALUES RETURNED

SS$_NORMAL	Service successfully completed.
SS$_ACCVIO	The message buffer, message buffer descriptor, device name string, or device name string descriptor cannot be read by the caller.
SS$_BADPARAM	The message length exceeds 16,350 bytes; the process's buffered I/O byte count limit (BYTLM) quota is insufficient; or the message length exceeds the value specified by the SYSGEN parameter MAXBUF.
SS$_DEVOFFLINE	The specified terminal is offline, has enabled PASSALL mode, or is not a terminal.
SS$_EXQUOTA	The process has exceeded its buffer space quota and has disabled resource wait mode with the Set Resource Wait Mode ($SETRWM) service.
SS$_INSFMEM	Insufficient system dynamic memory is available to complete the request and the process has disabled resource wait mode with the Set Resource Wait Mode ($SETRWM) service.
SS$_NONLOCAL	Warning. The device is on a remote node.
SS$_NOPRIV	The process does not have the privilege to broadcast messages.
SS$_NOSUCHDEV	Warning. The specified terminal does not exist, or it cannot receive the message.

$CNTREG Contract Program/Control Region

The Contract Program/Control Region service deletes a specified number of pages from the current end of the program or control region of a process's virtual address space. The deleted pages become inaccessible, and references to them cause access violations.

Note: Do not use the $CNTREG, or $CRETVA system services in conjunction with other user-written procedures and/or DIGITAL-supplied procedures (including run-time library procedures). These system services provide no means to communicate a change in virtual address space with other routines. DIGITAL recommends that you use either $EXPREG or the Run-time Library Procedure Allocate Virtual Memory (LIB$GET_VM) to get memory. When using $DELTVA, you should take care to only delete pages that you have specifically created.

FORMAT

SYS$CNTREG *pagcnt ,[retadr] ,[acmode] ,[region]*

RETURNS

VMS usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

Longword condition value. All system services return (by immediate value) a condition value in R0. Condition values that can be returned by this service are listed under "CONDITION VALUES RETURNED."

ARGUMENTS

pagcnt

VMS usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

Number of pages to be deleted from the current end of the program or control region. The **pagcnt** argument is a longword specifying this number.

retadr

VMS usage: **address_range**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

Starting and ending pages of the deleted area. The **retadr** argument is the address of a two-longword array to receive the virtual addresses of the starting page and ending page of the deleted area.

acmode

VMS usage: **access_mode**
type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

Access mode of the owner of the pages to be deleted. The **acmode** argument is a longword containing the access mode. The $PSLDEF macro defines the following symbols for the four access modes:

Symbol	Access mode
PSL$C_KERNEL	Kernel mode
PSL$C_EXEC	Executive mode
PSL$C_SUPER	Supervisor mode
PSL$C_USER	User mode

The most privileged access mode used is the access mode of the caller.

region

VMS usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

Indicator specifying which region of memory (P0 or P1) is to be contracted. The **region** argument is a longword containing the indicator. A value of 0 (the default) indicates that the program region (P0 region) is to be contracted, and a value of 1 indicates that the control region (P1 region) is to be contracted.

DESCRIPTION

If an error occurs while deleting pages, the **retadr** argument, if specified, indicates the range of pages that were successfully deleted before the error occurred. If no pages were deleted, both longwords in **retadr** contain a –1.

The $CNTREG service can delete pages only from the current end of the process's program or control region. To delete a specific range of pages in either region, use the Delete Virtual Address Space ($DELTVA) service.

CONDITION VALUES RETURNED

SS$_NORMAL	Service successfully completed.
SS$_ACCVIO	The **retadr** argument cannot be written by the caller.
SS$_ILLPAGCNT	The specified page count was less than 1.
SS$_PAGOWNVIO	A page in the specified range is owned by a more privileged access mode.

$CRELOG Create Logical Name

The Create Logical Name service inserts a logical name and its equivalence name into the process, group, or system logical name table. If the logical name already exists in the respective table, the new definition supersedes the old.

The Create Logical Name ($CRELNM) service supersedes the $CRELOG service. New programs should be written using $CRELNM, not $CRELOG, and old programs that use $CRELOG should be converted to use $CRELNM.

FORMAT

SYS$CRELOG *[tblflg] ,lognam ,eqlnam [,acmode]*

RETURNS

VMS usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

Longword condition value. All system services return (by value) a condition value in R0. Condition values that can be returned by this service are listed under "CONDITION VALUES RETURNED."

ARGUMENTS

tblflg

VMS usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

Logical name table into which the newly created logical name is to be placed. The **tblflg** argument is a longword value specifying this table. A value of 0 (the default) specifies the system table; 1, the group table; and 2, the process table.

lognam

VMS usage: **logical_name**
type: **character-coded text string**
access: **read only**
mechanism: **by descriptor–fixed length string descriptor**

Name of the logical name to be created. The **lognam** is the address of a character string descriptor pointing to the logical name string.

eqlnam

VMS usage: **logical_name**
type: **character-coded text string**
access: **read only**
mechanism: **by descriptor–fixed length string descriptor**

Equivalence name string for the logical name. The **eqlnam** is the address of a character string descriptor pointing to the equivalence name string.

acmode

VMS usage: **access_mode**
type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

Access mode to be associated with the logical name table entry. The **acmode** argument is a longword containing the access mode. The access mode only qualifies names in the process logical name table. The $PSLDEF macro defines the following symbols for the four access modes:

Symbol	Access mode
PSL$C_KERNEL	Kernel mode
PSL$C_EXEC	Executive mode
PSL$C_SUPER	Supervisor mode
PSL$C_USER	User mode

The most privileged access mode used is the access mode of the caller.

DESCRIPTION

The calling process must have the following:

- GRPNAM privilege to place an entry in the group logical name table
- SYSNAM privilege to place an entry in the system logical name table

Creation of logical names for the group and system logical name tables requires system dynamic memory.

Logical names can also be created from the command stream, with the DCL commands ASSIGN, DEFINE, ALLOCATE, and MOUNT.

CONDITION VALUES RETURNED

SS$_NORMAL	Service successfully completed. A new name was entered in the specified logical name table.
SS$_SUPERSEDE	Service successfully completed. A new equivalence name replaced a previous equivalence name in the specified logical name table.
SS$_ACCVIO	The logical name or equivalence name string or string descriptor cannot be read by the caller.
SS$_INSFMEM	Insufficient system dynamic memory is available to allocate a group or system logical name table entry, or the process has exceeded its limit for process logical name table entries.
SS$_IVLOGNAM	The logical name or equivalence name string has a length of 0 or has more than 255 characters.
SS$_IVLOGTAB	An invalid logical name table number was specified.
SS$_NOPRIV	The process does not have the privilege to place an entry in the specified logical name table.

$DELLOG Delete Logical Name

The Delete Logical Name service deletes a logical name and its equivalence name from the process, group, or system logical name table.

The Delete Logical Name ($DELLNM) service supersedes the $DELLOG service. New programs should be written using $DELLNM, not $DELLOG, and old programs that use $DELLOG should be converted to use $DELLNM.

FORMAT

SYS$DELLOG *[tblflg] [,lognam] [,acmode]*

RETURNS

VMS usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

Longword condition value. All system services return (by value) a condition value in R0. Condition values that can be returned by this service are listed under "CONDITION VALUES RETURNED."

ARGUMENTS

tblflg

VMS usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

Logical name table flag indicating the table in which the logical name to be deleted resides. The **tblflg** argument is a longword value. A value of 0 (the default) specifies the system table; 1 specifies the group table; and 2 specifies the process table.

lognam

VMS usage: **logical_name**
type: **character-coded text string**
access: **read only**
mechanism: **by descriptor–fixed length string descriptor**

Logical name to be deleted. The **lognam** argument is the address of a character string descriptor pointing to the logical name string. If **lognam** is not specified, $DELLOG deletes all logical names that the process is privileged to delete, in the specified table. The maximum length of a logical name is 255 characters.

acmode

VMS usage: **access_mode**
type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

Access mode associated with the process logical name table entry. This argument is used only for deleting names from the process logical name table. The **acmode** argument is a longword containing the access mode.

The most privileged access mode used is the access mode of the caller; only the logical name entered at the resulting access mode or a less privileged access mode is deleted.

DESCRIPTION

The calling process must have the following:

- GRPNAM privilege to delete a logical name from the group logical name table
- SYSNAM privilege to delete a logical name from the system logical name table

Deletion of a logical name from the group or system table returns storage to system dynamic memory. Deletion of a logical name from the process table returns storage to the control region of the process's virtual address space.

Logical names can be deleted from the command stream with the DEASSIGN command.

Names in the process logical name table that were created from user mode are automatically deleted at image exit.

CONDITION VALUES RETURNED

SS$_NORMAL	Service successfully completed.
SS$_ACCVIO	The **lognam** argument or the logical name string cannot be read by the caller.
SS$_IVLOGNAM	The logical name string has a length of 0 or has more than 255 characters. Note that, prior to Version 4 of VMS, SS$_IVLOGNAM was returned when the length of a logical name exceeded 63 characters.
SS$_IVLOGTAB	An invalid logical name table number was specified.
SS$_NOLOGNAM	Either (1) the specified logical name does not exist in the specified logical name table, or (2) the specified logical name exists in the process logical name table but the entry was made from a more privileged access mode.
SS$_NOPRIV	The process does not have the privilege to delete an entry from the specified logical name table.

$GETCHN Get I/O Channel Information

The Get I/O Channel Information service returns information about a device to which the calling process has assigned an I/O channel. The information returned consists of either primary or secondary device characteristics, or both.

The Get Device/Volume Information ($GETDVI) service supersedes the $GETCHN service. New programs should be written using $GETDVI instead of $GETCHN, and old programs that use $GETCHN should be converted to use $GETDVI or $GETDVIW.

FORMAT

SYS$GETCHN *chan ,[prilen] ,[pribuf] ,[scdlen] ,[scdbuf]*

RETURNS

VMS usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

Longword condition value. All system services return (by value) a condition value in R0. Condition values that can be returned by this service are listed under "CONDITION VALUES RETURNED."

ARGUMENTS

chan

VMS usage: **channel**
type: **word (unsigned)**
access: **read only**
mechanism: **by value**

Number of the I/O channel assigned to the device. The **chan** argument is a longword containing this number.

prilen

VMS usage: **word_unsigned**
type: **word (unsigned)**
access: **write only**
mechanism: **by reference**

Length in bytes of the primary device characteristics that $GETCHN returns to the caller. The **prilen** argument is the address of a word into which $GETCHN writes the length.

pribuf

VMS usage: **char_string**
type: **character-coded text string**
access: **write only**
mechanism: **by descriptor–fixed length string descriptor**

Buffer into which $GETCHN writes the primary device characteristics. The **pribuf** argument is the address of a character string descriptor pointing to this buffer.

If **pribuf** is specified as 0 (an address of 0), $GETCHN interprets this to mean that no buffer is specified. This is the default.

scdlen

VMS usage: **word_unsigned**
type: **word (unsigned)**
access: **write only**
mechanism: **by reference**

Length in bytes of the secondary device characteristics that $GETCHN returns to the caller. The **scdlen** argument is the address of a word into which $GETCHN writes the length.

scdbuf

VMS usage: **char_string**
type: **character-coded text string**
access: **write only**
mechanism: **by descriptor–fixed length string descriptor**

Buffer into which $GETCHN writes the secondary device characteristics. The **scdbuf** argument is the address of a character string descriptor pointing to this buffer.

If **scdbuf** is specified as 0 (an address of 0), $GETCHN interprets this to mean that no buffer is specified. This is the default.

DESCRIPTION

The channel number specified in the **chan** argument must correspond to an assigned channel.

In most cases, the two sets of characteristics (primary and secondary) returned by $GETCHN are identical. However, the two sets provide different information in the following cases:

- If the device has an associated mailbox, the primary characteristics are those of the assigned device and the secondary characteristics are those of the associated mailbox.
- If the device is a spooled device, the primary characteristics are those of the intermediate device and the secondary characteristics are those of the spooled device.
- If the device represents a logical link on the network, the secondary characteristics contain information about the link.

Note also that the Get I/O Device Information ($GETDEV) service returns the same information as the $GETCHN service. Both services return information in a user-supplied buffer. Symbolic names defined in the $DIBDEF macro represent offsets from the beginning of the buffer. The length of the buffer is defined in the constant DIB$K_LENGTH, and the field offset names, lengths, and contents are listed below.

Field Name	Length (bytes)	Contents
DIB$L_DEVCHAR	4	Device characteristics
DIB$B_DEVCLASS	1	Device class
DIB$B_DEVTYPE	1	Device type
DIB$B_SECTORS	1	Number of sectors per track (disk)
DIB$B_TRACKS	1	Number of tracks per cylinder (disk)
DIB$W_CYLINDERS	2	Number of cylinders on the volume (disk)
DIB$W_DEVBUFSIZ	2	Device buffer size
DIB$L_DEVDEPEND	4	Device-dependent information
DIB$L_MAXBLOCK	4	Number of logical blocks on the volume (disk)
DIB$W_UNIT	2	Unit number
DIB$W_DEVNAMOFF	2	Offset to device name string
DIB$L_PID	4	Process identification of device owner
DIB$L_OWNUIC	4	UIC of device owner
DIB$W_VPROT	2	Volume protection mask
DIB$W_ERRCNT	2	Error count
DIB$L_OPCNT	4	Operation count
DIB$W_VOLNAMOFF	2	Offset to volume label string
DIB$W_RECSIZ	2	Blocked record size; valid for magnetic tapes when DIB$W_VOLNAMOFF is nonzero

The device name string and volume label string are returned in the buffer as counted ASCII strings and must be located by using their offsets from the beginning of the buffer.

Any fields not applicable to a particular device are returned as zeros.

CONDITION VALUES RETURNED

SS$_NORMAL	Service successfully completed.
SS$_BUFFEROVF	Service successfully completed. The device information returned overflowed the buffer(s) provided and has been truncated.
SS$_ACCVIO	A buffer descriptor cannot be read by the caller, or a buffer or buffer length cannot be written by the caller.
SS$_IVCHAN	An invalid channel number was specified, that is, a channel number of 0 or a number larger than the number of channels available.
SS$_NOPRIV	The specified channel is not assigned or was assigned from a more privileged access mode.

$GETDEV Get I/O Device Information

The Get I/O Device Information service returns information about an I/O device; the calling process need not have assigned a channel to the device. The information returned consists of either primary or secondary device characteristics, or both.

The Get Device/Volume Information ($GETDVI) service supersedes the $GETDEV service. New programs should be written using $GETDVI instead of $GETDEV, and old programs that use $GETDEV should be converted to use $GETDVI or $GETDVIW.

FORMAT

SYS$GETDEV *devnam ,[prilen] ,[pribuf] ,[scdlen] ,[scdbuf]*

RETURNS

VMS usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

Longword condition value. All system services return (by value) a condition value in R0. Condition values that can be returned by this service are listed under "CONDITION VALUES RETURNED."

ARGUMENTS

devnam

VMS usage: **device_name**
type: **character-coded text string**
access: **read only**
mechanism: **by descriptor–fixed length string descriptor**

Name of the device about which information is to be returned. The **devnam** argument is the address of a character string descriptor pointing to this device name string.

prilen

VMS usage: **Word (unsigned)**
type: **word (unsigned)**
access: **write only**
mechanism: **by reference**

Length in bytes of the primary device characteristics that $GETDEV returns to the caller. The **prilen** argument is the address of a word into which $GETDEV writes the length.

pribuf

VMS usage: **char_string**
type: **character-coded text string**
access: **write only**
mechanism: **by descriptor–fixed length string descriptor**

Buffer into which $GETDEV writes the primary device characteristics. The **pribuf** argument is the address of a character string descriptor pointing to this buffer.

If **pribuf** is specified as 0 (an address of 0), $GETDEV interprets this to mean that no buffer is specified. This is the default.

scdlen

VMS usage: **word_unsigned**
type: **word (unsigned)**
access: **write only**
mechanism: **by reference**

Length in bytes of the secondary device characteristics that $GETDEV returns to the caller. The **scdlen** argument is the address of a word into which $GETDEV writes the length.

scdbuf

VMS usage: **char_string**
type: **character-coded text string**
access: **write only**
mechanism: **by descriptor–fixed length string descriptor**

Buffer into which $GETDEV writes the secondary device characteristics. The **scdbuf** argument is the address of a character string descriptor pointing to this buffer.

If **scdbuf** is specified as 0 (an address of 0), $GETDEV interprets this to mean that no buffer is specified. This is the default.

DESCRIPTION

In most cases, the two sets of characteristics (primary and secondary) returned by $GETDEV are identical. However, the two sets provide different information in the following cases:

- If the device has an associated mailbox, the primary characteristics are those of the assigned device and the secondary characteristics are those of the associated mailbox.
- If the device is a spooled device, the primary characteristics are those of the intermediate device and the secondary characteristics are those of the spooled device.
- If the device represents a logical link on the network, the secondary characteristics contain information about the link.

The Get I/O Channel Information ($GETCHN) service returns the same information as the $GETDEV service and returns it in the same format. Both services return information in a user-supplied buffer. The $DIBDEF macro defines symbolic names for the length and contents of this buffer; refer to the Description section of the $GETCHN service for a list of these symbolic names.

CONDITION VALUES RETURNED

SS$_NORMAL	Service successfully completed.
SS$_BUFFEROVF	Service successfully completed. The device information returned overflowed the buffer(s) provided and has been truncated.
SS$_ACCVIO	A buffer descriptor cannot be read by the caller, or a buffer or buffer length cannot be written by the caller.
SS$_IVDEVNAM	No device name was specified, or the device name string has invalid characters.
SS$_IVLOGNAM	The device name string has a length of 0 or has more than 63 characters.
SS$_NONLOCAL	Warning. The device is on a remote system.
SS$_NOSUCHDEV	Warning. The specified device does not exist on the host system.

$INPUT Queue Input Request and Wait for Event Flag

$INPUT is a simplified form of the Queue I/O Request and Wait for Event Flag ($QIOW) service. It queues a virtual input operation using the IO$_READVBLK function code and waits for I/O completion.

$INPUT is not a system service, but rather a macro that may be invoked only by the VAX MACRO language.

DIGITAL no longer recommends the use of $INPUT, even by VAX MACRO users; the $QIO or $QIOW services should be used instead.

FORMAT

SYS$INPUT *chan ,length ,buffer [,iosb] [,efn]*

RETURNS

type:
access:
mechanism:

Longword condition value. All system services return (by value) a condition value in R0. Condition values that can be returned by this service are listed under "CONDITION VALUES RETURNED."

ARGUMENTS

chan
Number of the I/O channel assigned to the device from which input is to be read.

length
Length of the input buffer.

buffer
Address of the input buffer.

iosb
Address of a quadword I/O status block.

efn
Number of the event flag to be set when the request is complete. The default is event flag 0.

DESCRIPTION

The $INPUT macro has only one form. Arguments must be specified as for the $name_S macro form, but "_S" must not be included in the macro call.

CONDITION VALUES RETURNED

Any condition values returned by $QIO

$OUTPUT Queue Output Request and Wait for Event Flag

$OUTPUT is a simplified form of the Queue I/O Request and Wait for Event Flag ($QIOW) service. It queues a virtual output operation using the IO$_WRITEVBLKBLK function code and waits for I/O completion.

$OUTPUT is not a system service, but rather a macro that may be invoked only by the VAX MACRO language.

DIGITAL no longer recommends the use of $OUTPUT, even by VAX MACRO users; the $QIO or $QIOW services should be used instead.

FORMAT

SYS$OUTPUT *chan ,length ,buffer [,iosb] [,efn]*

RETURNS

type:
access:
mechanism:

Longword condition value. All system services return (by value) a condition value in R0. Condition values that can be returned by this service are listed under "CONDITION VALUES RETURNED."

ARGUMENTS

chan
Number of the I/O channel assigned to the device to which output is to be written.

length
Length of the output buffer.

buffer
Address of the output buffer.

iosb
Address of the quadword I/O status block.

efn
Number of the event flag to be set when the request is complete. The default is event flag 0.

DESCRIPTION

The $OUTPUT macro has only one form. Arguments must be specified as for the $name_S macro form, but "_S" must not be included in the macro call.

The $OUTPUT macro supplies a P4 value of hexadecimal 20 to the $QIOW service. For output to a terminal, this value is a carriage control specifier indicating the following sequence: line feed, print buffer contents, carriage return.

CONDITION VALUES RETURNED

Same as those for $QIO

$SNDACC Send Message to Accounting Manager

The Send Message to Accounting Manager service controls accounting log activity and allows a process to write an arbitrary data message into the accounting log file.

The Send to Job Controller ($SNDJBC) service supersedes the $SNDACC service. New code should be written using $SNDJBC (or $SNDJBCW) instead of $SNDACC, and old code using $SNDACC should be converted to use $SNDJBC or $SNDJBCW.

FORMAT

SYS$SNDACC *msgbuf ,[chan]*

RETURNS

VMS usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

Longword condition value. All system services return (by value) a condition value in R0. Condition values that can be returned by this service are listed under "CONDITION VALUES RETURNED."

ARGUMENTS

msgbuf

VMS usage: **char_string**
type: **character-coded text string**
access: **read only**
mechanism: **by descriptor–fixed length string descriptor**

Message buffer specifying the message type and message. The **msgbuf** argument is the address of a character string descriptor pointing to the message buffer.

The first word in the message buffer must specify a message type. These message types have symbolic names, which are defined by the $ACCDEF macro. Some message types require that data follow the message type code in the message buffer, while other message types require no data at all. The following lists each message type and describes the nature of the data, if any, that must follow that type.

$SNDACC Message Types

ACC$K_INSMESG

This message type directs $SNDACC to insert a message in the accounting log file. Following the message type in the buffer is the message text itself. $SNDACC precedes the message with a default header when the message is written to the accounting log file.

ACC$K_NEWFILE

This message type directs $SNDACC to close the current accounting log file and to open a new accounting log file. OPER privilege is required to issue this request. No data follows the message type in the buffer.

ACC$K_ENABACC

This message type enables accounting for all types of jobs. OPER privilege is required to issue this request. No data follows the message type in the buffer.

ACC$K_DISAACC

This message type disables accounting for all types of jobs. OPER privilege is required to issue this request. No data follows the message type in the buffer.

ACC$K_ENABSEL

This message type enables accounting for certain types of jobs. OPER privilege is required to issue this request. In the buffer, the message type is followed by one or more 1-byte job type codes that define the types of job for which accounting is to be enabled. The list of job type codes must be terminated with a byte containing 0. The following job type codes are defined by the $ACCDEF macro:

Type Code	Job Type
ACC$K_BATTRM	Batch job
ACC$K_INSMSG	User message
ACC$K_INTTRM	Interactive job
ACC$K_LOGTRM	Login failure
ACC$K_PRCTRM	Noninteractive process, subprocess, or detached process
ACC$K_PRTJOB	Print job

ACC$K_DISASEL

This message type disables accounting for certain types of jobs. OPER privilege is required to issue this request. In the buffer, the message type is followed by one or more 1-byte job type codes that define the types of job for which accounting is to be disabled. The list of job type codes must be terminated with a byte containing 0. The list of job type codes appears under the description of the ACC$K_ENABSEL message type code.

chan

VMS usage: **channel**
type: **word (unsigned)**
access: **read only**
mechanism: **by value**

Channel number of the mailbox to which $SNDACC writes the completion status of the operation. The **chan** argument is a longword containing this number.

If the **chan** argument is not specified or is specified as 0, $SNDACC does not return the completion status. The completion status is lost if insufficient buffer space exists, if the message is too long, or if the mailbox no longer exists when $SNDACC attempts to return the status.

$SNDACC

The $SNDACC returns the completion status to the mailbox in the following format:

Bits	Description
0–15	In this word $SNDACC writes the code MSG$_ACCRSP, indicating that this mailbox has been written to by $SNDACC. This code is defined by the $MSGDEF macro.
16–31	0
32–63	Completion status code. These codes are defined by the $JBCMSGDEF macro and are listed under "Condition Values Returned in the Mailbox" at the end of the $SNDACC description.

DESCRIPTION

The calling process must have OPER privilege to: (1) create a new log; or (2) enable or disable accounting.

The $SNDACC service requires system dynamic memory.

The accounting log file is located on the system disk in SYS$MANAGER:ACCOUNTNG.DAT. The file is sequentially organized and contains variable-length records.

The general procedure for writing a call to this service is as follows:

1. Construct the message buffer and place its final length in the first word of the buffer descriptor.
2. Call the $SNDACC service.
3. Check the status code returned in R0 for successful completion.
4. Issue a read request to the mailbox specified, if any.
5. When the read completes, check the status code returned in the mailbox for successful completion.

By default, the system writes a record into the accounting log file whenever a job terminates. Termination records are written for interactive users, batch jobs, noninteractive processes, login failures, and print jobs.

The $SNDACC service allows users to write additional data into the accounting log and allows privileged users to disable or enable all accounting for particular types of jobs.

CONDITION VALUES RETURNED

SS$_NORMAL	Successful completion.
SS$_ACCVIO	The message buffer or buffer descriptor cannot be read by the caller.
SS$_BADPARAM	The specified message has a length of 0 or has more than 196 characters.
SS$_DEVNOTMBX	The channel specified is not assigned to a mailbox.
SS$_INSFMEM	Insufficient system dynamic memory is available to complete the service.

SS$_IVCHAN	An invalid channel number was specified, that is, a channel number of 0 or a number larger than the number of channels available.
SS$_NOPRIV	The caller does not have write access to the specified mailbox.

CONDITION VALUES RETURNED IN THE MAILBOX

JBC$_NORMAL	Successful completion.
JBC$_NOOPER	The caller does not have the necessary OPER privilege for the requested operation.
JBC$_INVFUNCOD	The specified message type code is invalid.
JBC$_INVITMCOD	The specified job type code is invalid.
JBC$_INVMSGBUF	The message buffer is not large enough to contain all required fields.
JBC$_INVPARLEN	The message text is more than 255 characters.

EXAMPLE

```
        $ACCDEF
ACCMSG: .WORD   0                       ; mailbox channel number
MESSAGE:                                ; character string descriptor
        .WORD   ENDRQ-REQUEST           ; size of request
        .WORD   0
        .ADDRESS -                      ; address of request
                REQUEST
REQUEST:                                ; message buffer for $SNDACC
        .WORD   ACC$K_ENABSEL           ; selectively enable:
        .BYTE   ACC$K_BATTRM            ; batch jobs
        .BYTE   ACC$K_PRTJOB            ; print jobs
        .BYTE   0                       ; end of message buffer
ENDRQ:
          .
          .
          .
        $CREMBX_S -                     ; create mailbox for message
                chan=ACCMSG
          .
          .
          .
        $SNDACC_S -
                msgbuf=MESSAGE, -       ; address of descriptor
                chan=ACCMSG             ; mailbox channel for data
```

The above example shows a segment of a program used to selectively enable accounting for batch jobs and print jobs.

$SNDSMB Send Message to Symbiont Manager

The Send Message to Symbiont Manager service allows a user to create and manage queues, as well as the jobs in those queues.

The Send to Job Controller ($SNDJBC) service supersedes the $SNDSMB service. New code should be written using $SNDJBC (or $SNDJBCW) instead of $SNDSMB, and old code using $SNDSMB should be converted to use $SNDJBC or $SNDJBCW.

FORMAT

SYS$SNDSMB *msgbuf ,[chan]*

RETURNS

VMS usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

Longword condition value. All system services return (by value) a condition value in R0. Condition values that can be returned by this service are listed under "CONDITION VALUES RETURNED."

ARGUMENTS

msgbuf

VMS usage: **char_string**
type: **character-coded text string**
access: **read only**
mechanism: **by descriptor–fixed length string descriptor**

Address of a character string descriptor that points to the message buffer. The user constructs a message buffer for each $SNDSMB request. The first 2-byte field of the buffer specifies the request type; additional fields specify other required information; the last field specifies other optional information.

The size and content of the message buffer varies depending on the $SNDSMB request. The Description section contains a list of $SNDSMB request types and, for each type, shows the content and format of the message buffer including allowable options. The Description section also contains a list describing each option.

chan

VMS usage: **channel**
type: **word (unsigned)**
access: **read only**
mechanism: **by value**

Number of the channel assigned to the mailbox that is to receive the reply from $SNDSMB. If no channel number is specified or if it is specified as 0 (the default), $SNDSMB does not reply.

If the **chan** argument is specified, the symbiont manager returns one quadword of information to the mailbox. This information is formatted as follows:

Bits	Contents
0–15	MSG$_SMBRSP. This name, which is defined by the $MSGDEF macro, indicates that the message is from the symbiont manager.
16–31	The job entry number (or jobid). The symbiont manager assigns a jobid to each batch or print job.
32–63	Status code. This status code describes the actual results of the symbiont operation. These status codes are defined by the $JBCMSGDEF macro, which is located in SYS$LIBRARY:LIB.MLB. A list of these status codes appears under "Condition Values Returned in the Mailbox". Note that, in contrast to these status codes, the status codes returned in R0 describe the call to $SNDSMB (whether it was successful, ill-formed, and so on).

If the mailbox cannot handle the message because either (1) there is insufficient buffer space, (2) the message is too long, or (3) the mailbox no longer exists when the reply is sent, then the response is lost.

DESCRIPTION

The Send Message To Symbiont Manager service requires system dynamic memory.

The general procedure for using this service is as follows:

1. Construct the message buffer and place its final length in the first word of the buffer descriptor.
2. Issue the $SNDSMB service.
3. Check the return status code from the service to ensure successful completion.
4. Issue a read request to the specified mailbox, if any.
5. When the read completes, check that the operation was successfully performed.

$SNDSMB Request Types and Message Buffer Formats

This section lists each $SNDSMB request type, describes the operation it designates, shows the content and format of the message buffer for the request type, and lists the options that may be specified with the request type.

The format of the message buffer for each request type is described in terms of required and optional fields. Some of the items for required fields are used for several request types; rather than repeat the syntactical description of a required item for each request type, the syntactical description is given once here:

Queue name	The total length of this field must be 16 bytes. The first byte contains the length of the queue name string (maximum 15 characters). Successive bytes contain the queue name string, one character per byte. Unused bytes are filled with any characters to bring the total length of the field to 16 bytes.
Device name	The total length of this field must be 16 bytes. The contents of this field is actually supplied by RMS; the user need only copy to the message buffer 16 bytes beginning at offset NAM$T_DVI of the RMS Name Block (NAM).
File ID	The total length of this field must be 6 bytes. The contents of this field is actually supplied by RMS; the user need only copy to the message buffer 6 bytes beginning at offset NAM$W_FID of the RMS Name Block (NAM).
Dir ID	The total length of this field must be 6 bytes. The contents of this field is actually supplied by RMS; the user need only copy to the message buffer 6 bytes beginning at offset NAM$W_DID of the RMS Name Block (NAM).
File name	The total length of this field must be 20 bytes. It is not used.
Job name	The total length of this field must be 10 bytes. The first byte contains the length of the job name string (maximum 9 characters). Successive bytes contain the job name string, one character per byte. Unused bytes are filled with any characters to bring the total length of the field to 10 bytes.

If an option is specified in the message buffer, it must be specified after all required fields. If the option requires a data value, that value must immediately follow the option in the message buffer.

After all options and their data values (if any) have been specified in the message buffer, an option code of 0 may be entered in the message buffer. The zero option code indicates the end of all options (and the end of the message buffer).

SMR$K_ABORT

This request stops the current job on a specified queue and starts the next job in that queue.

Required fields in the message buffer:

SMR$K_ABORT	2-byte request type
Queue name	16-byte name of the queue whose current job is to be stopped

Optional contents of the message buffer (valid only for output queues):

SMO$K_REQDEST SMO$K_REQPRIO SMO$K_REQUEUE

SMR$K_ADDFIL

This request adds a file to a job that was created by the SMR$K_CREJOB request.

Required fields in the message buffer:

SMR$K_ADDFIL	2-byte request type
Device name	16-byte device name that identifies the device on which the file resides
File ID	6-byte file identification of the file
Dir ID	6-byte directory identification of the file
File name	20-byte unused field

Optional contents of the message buffer:

SMO$K_COPIES	SMO$K_BRSTPAG	SMO$K_DELETE
SMO$K_DOUBLE	SMO$K_FILESIZ	SMO$K_FLAGPAG
SMO$K_NOBRSTPAG	SMO$K_NOFEED	SMO$K_NOFLAGPAG
SMO$K_PAGCNT	SMO$K_PAGHDR	

SMR$K_ALTER

This request changes the attributes of a previously queued job and then requeues the job.

Required fields in the message buffer:

SMR$K_ALTER	2-byte request type
Queue name	16-byte queue name that identifies the queue on which the job is queued
Job ID	2-byte job entry number; this identifier was returned by $SNDSMB on a previous call when the job, which is now to be altered, was initially queued

Optional contents of the message buffer:

SMO$K_CPULIM	SMO$K_DQCHAR	SMO$K_DESTQUE
SMO$K_FORMTYPE	SMO$K_HOLD	SMO$K_JOBCOPY
SMO$K_JOBNAME	SMO$K_JOBPRI	SMO$K_LOWER
SMO$K_NOCPULM	SMO$K_NOLOWER	SMO$K_NOWSDFT
SMO$K_ NOWSEXTNT	SMO$K_NOWSQUO	SMO$K_RLSTIM
SMO$K_WSDEFLT	SMO$K_WSEXTNT	SMO$K_WSQUOTA

SMR$K_CLSJOB

This request closes the current job and enters it on the queue specified in the SMR$K_CREJOB request. SMR$K_CLSJOB works in tandem with SMR$K_CREJOB. For example, the user creates a job with the SMR$K_CREJOB request, adds files to the job with the SMR$K_ADDFIL request, and closes the job with the SMR$K_CLSJOB request.

Required fields in the message buffer:

SMR$K_CLSJOB	2-byte request type

No options are allowed.

SMR$K_CREJOB

This request creates a job on a specified queue. It is used to create a job consisting of a number of files and is used in tandem with SMR$K_ADDFIL and SMR$K_CLSJOB. See the description of SMR$K_CLSJOB above.

Required fields in the message buffer:

SMR$K_CREJOB	2-byte request type
Queue name	16-byte name of the queue that identifies the queue on which the job is to be queued

Optional contents of the message buffer:

SMO$K_CPULIM	SMO$K_DQCHAR	SMO$K_FORMTYPE
SMO$K_HOLD	SMO$K_JOBCOPY	SMO$K_JOBNAME
SMO$K_JOBPRI	SMO$K_KEEPLOG	SMO$K_LOGFNAM
SMO$K_LOGQNAM	SMO$K_LOWER	SMO$K_NOCPULM
SMO$K_NOKEEPLOG	SMO$K_NOLOG	SMO$K_NOLOWER
SMO$K_NONOTFY	SMO$K_NOSPOOL	SMO$K_NOTIFY
SMO$K_NOWSDFT	SMO$K_ NOWSEXTNT	SMO$K_NOWSQUO
SMO$K_PARAMS	SMO$K_RLSTIM	SMO$K_WSDEFLT
SMO$K_WSEXTNT	SMO$K_WSQUOTA	

SMR$K_DELETE

This request deletes a queue.

Required fields in the message buffer:

SMR$K_DELETE	2-byte request type
Queue name	16-byte name of the queue to be deleted

No options are allowed.

SMR$K_ENTER

This request enters a single file in a queue. Use SMR$K_CREJOB to enter more than one file in a queue.

Required fields in the message buffer:

SMR$K_ENTER	2-byte request type
Queue name	16-byte name that identifies the queue in which the file is to be queued
Device name	16-byte device name that identifies the device on which the file resides
File ID	6-byte file identification of the file

Dir ID	6-byte directory identification of the file (required only if the file is to be deleted after processing)
File name	20-byte unused field

Optional contents of the message buffer:

SMO$K_BRSTPAG	SMO$K_COPIES	SMO$K_CPULIM
SMO$K_DELETE	SMO$K_DOUBLE	SMO$K_DQCHAR
SMO$K_FILESIZ	SMO$K_FLAGPAG	SMO$K_FORMTYPE
SMO$K_HOLD	SMO$K_JOBCOPY	SMO$K_JOBNAME
SMO$K_JOBPRI	SMO$K_KEEPLOG	SMO$K_LOGFNAM
SMO$K_LOGQNAM	SMO$K_LOWER	SMO$K_NOBRSTPAG
SMO$K_NOCPULM	SMO$K_NOFEED	SMO$K_NOFLAGPAG
SMO$K_NOKEEPLOG	SMO$K_NOLOG	SMO$K_NOLOWER
SMO$K_NONOTFY	SMO$K_NOSPOOL	SMO$K_NOTIFY
SMO$K_NOWSDFT	SMO$K_NOWSEXTNT	SMO$K_NOWSQUO
SMO$K_PAGCNT	SMO$K_PAGHDR	SMO$K_PARAMS
SMO$K_RLSTIM	SMO$K_WSDEFLT	SMO$K_WSEXTNT
SMO$K_WSQUOTA		

SMR$K_INITIAL

This request initializes or reinitializes a queue.

Required fields in the message buffer:

SMR$K_INITIAL	2-byte request type
Queue name	16-byte name of the queue to be initialized or reinitialized

Optional contents of the message buffer:

SMO$K_ CURDQCHAR	SMO$K_CURFORM	SMO$K_DCPULM
SMO$K_DEFBRST	SMO$K_DEFFLAG	SMO$K_DETJOB
SMO$K_DISWAP	SMO$K_GENDEV	SMO$K_GENPRT
SMO$K_INIPRI	SMO$K_JOBLIM	SMO$K_MCPULM
SMO$K_NODCPULM	SMO$K_NODEFBRST	SMO$K_NODEFFLAG
SMO$K_NODISWAP	SMO$K_NOGENDEV	SMO$K_NOGENPRT
SMO$K_NOMCPULM	SMO$K_NOTRMDEV	SMO$K_NOWSDFLT
SMO$K_NOWSQUTA	SMO$K_NOWSXTNT	SMO$K_SMBNAME
SMO$K_TRMDEV	SMO$K_WSDFLT	SMO$K_WSQUTA
SMO$K_WSXTANT		

SMR$K_MERGE

This request deletes jobs from one queue (the source queue) and requeues them in another queue (the destination queue).

Required fields in the message buffer:

SMR$K_MERGE	2-byte request type
Destination queue name	16-byte name that identifies the queue to which the jobs are to be requeued
Source queue name	16-byte name that identifies the queue from which the jobs are to be deleted

No options are allowed.

SMR$K_PAUSE

This request pauses the execution of jobs in a specified queue.

Required fields in the message buffer:

SMR$K_PAUSE	2-byte request type
Queue name	16-byte name of the queue that is to be paused

No options are allowed.

SMR$K_REDIRECT

This request assigns a logical queue (the source queue) to an execution queue (the destination queue).

Required fields in the message buffer:

SMR$K_REDIRECT	2-byte request type
Destination queue name	16-byte name that identifies the execution queue to which a logical queue is to be assigned
Source queue name	16-byte name that identifies the logical queue which is to be assigned to the execution queue; if the source queue name field contains all binary zeros, the request revokes a previous assignment of the queue specified in the destination queue name field

No options are allowed.

SMR$K_RELEASE

This request releases a job that was put on hold by specifying the SMO$K_HOLD option with either the SMR$K_CREJOB or SMR$K_ENTER request types.

Required fields in the message buffer:

SMR$K_RELEASE	2-byte request type
Queue name	16-byte queue name field that identifies the queue in which the job is held
Job ID	2-byte job entry number; this identifier was returned by $SNDSMB on a previous call when the job, which is now to be released, was initially put on hold

No options are allowed.

SMR$K_RMVJOB

This request removes a job from a queue.

Required fields in the message buffer:

SMR$K_RMVJOB	2-byte request type
Queue name	16-byte queue name that identifies the queue from which to remove the job
Job ID	2-byte job entry number; this identifier was returned by $SNDSMB on a previous call when the job, which is now to be removed, was initially queued

No options are allowed.

SMR$K_START

This request enables printing on a device, resumes printing on a paused device, restarts printing on a stopped device, or starts a batch queue.

Required fields in the message buffer:

SMR$K_START	2-byte request type
Queue name	16-byte name of the queue to be started

Optional contents of the message buffer:

SMO$K_ CURDQCHAR	SMO$K_CURFORM	SMO$K_DCPULM
SMO$K_DEFBRST	SMO$K_DEFFLAG	SMO$K_DETJOB
SMO$K_DISWAP	SMO$K_GENDEV	SMO$K_GENPRT
SMO$K_INIPRI	SMO$K_JOBLIM	SMO$K_MCPULM
SMO$K_NEXTJOB	SMO$K_NODCPULM	SMO$K_NODEFBRST
SMO$K_NODEFFLAG	SMO$K_NOGENDEV	SMO$K_NOGENPRT
SMO$K_NOMCPULM	SMO$K_NOTRMDEV	SMO$K_NOWSDFLT
SMO$K_NOWSQUTA	SMO$K_NOWSXTNT	SMO$K_SMBNAME
SMO$K_SPCCNT	SMO$K_TOPOFILE	SMO$K_TRMDEV
SMO$K_WSDFLT	SMO$K_WSQUTA	SMO$K_WSXTANT

SMR$K_STOP

This request stops the execution of jobs in a queue.

Required fields in the message buffer:

SMR$K_STOP	2-byte request type
Queue name	16-byte name of the queue to be stopped

No options are allowed.

SMR$K_SYNCJOB

This request waits for a job to complete and then returns the 8-byte completion message. The status code contained in the second longword of the completion message is the completion status of the job.

Required fields in the message buffer (only one of either the job id or job name fields is required; if both are specified, the job id field is used):

SMR$K_SYNCJOB	2-byte request type
Queue name	16-byte name of the queue in which the job is queued
Job ID	2-byte job entry number that identifies the job for which to await completion; this identifier was returned by $SNDSMB on a previous call when the job was initially queued
Job name	10-byte job name

$SNDSMB Options

The following lists each $SNDSMB option. If an option requires that a data value also be specified, the format of the required data is described.

If an option is specified in the message buffer, it must be specified after all required fields. If the option requires a data value, that value must be specified in the message buffer immediately following its associated option.

After all options and their data values (if any) have been specified in the message buffer, an option code of 0 may be entered in the message buffer. The zero option code indicates the end of all options (and the end of the message buffer).

SMO$K_BRSTPAG
SMO$K_NOBRSTPAG

SMO$K_BRSTPAG specifies that a burst page be printed.

SMO$K_NOBRSTPAG specifies that a burst page not be printed.

No data value is required.

SMO$K_COPIES

SMO$K_COPIES specifies the number of copies of the file to be printed. The required 1-byte data field specifies the desired number.

SMO$K_CPULIM
SMO$K_NOCPULIM

SMO$K_CPULIM specifies the CPU time limit for a batch job. The required data field is an unsigned longword containing the desired number of 10 millisecond units.

SMO$K_NOCPULIM specifies that no CPU time limit is to be applied to the batch job. No data field is required.

SMO$K_CURDQCHAR

SMO$K_CURDQCHAR specifies the current queue characteristics. The required 16-byte data field contains a 128-bit mask. Each bit corresponds to a queue characteristic; set bits indicate that the corresponding characteristic is desired.

SMO$K_CURFORM

SMO$K_CURFORM specifies the form number currently on the printer. The required 1-byte data field specifies this form number.

SMO$K_DCPULM
SMO$K_NODCPULM

SMO$K_DCPULM specifies the default CPU time limit for batch jobs originating from a queue. The required data field is an unsigned longword

containing the desired number of 10 millisecond units. This number must be less than or equal to the maximum CPU time limit (specified by SMO$K_MCPULM).

SMO$K_NODCPULM specifies that no default CPU time limit is to be applied to batch jobs originating from a queue. No data field is required.

SMO$K_DEFBRST
SMO$K_NODEFBRST

SMO$K_DEFBRST specifies that by default a printing queue should print a burst page. No data field is required.

SMO$K_NODEFBRST specifies that by default a printing queue should not print a burst page. No data field is required.

SMO$K_DEFFLAG
SMO$K_NODEFFLAG

SMO$K_DEFFLAG specifies that by default a printing queue should print a flag page. No data field is required.

SMO$K_NODEFFLAG specifies that by default a printing queue should not print a flag page. No data field is required.

SMO$K_DELETE

SMO$K_DELETE specifies that a file be deleted after printing. The DIR ID field of the request is required.

SMO$K_DESTQUE

SMO$K_DESTQUE specifies the name of a new queue in which to put a job. The required data field is a counted string containing the name of the queue.

SMO$K_DETJOB

SMO$K_DETJOB specifies that a queue is defined as a batch queue. No data field is required.

SMO$K_DISWAP
SMO$K_NODISWAP

SMO$K_DISWAP disables the swapping of all batch jobs in a queue. No data field is required.

SMO$K_NODISWAP enables the swapping of all batch jobs in a queue. No data field is required.

SMO$K_DOUBLE

SMO$K_DOUBLE specifies that all print jobs be double-spaced. No data field is required.

SMO$K_DQCHAR

SMO$K_DQCHAR specifies characteristics that a device queue must have before a job in it can be dequeued. The required 16-byte data field is a 128-bit bit mask, where each bit corresponds to a characteristic.

SMO$K_FILESIZ

SMO$K_FILESIZ specifies the size of a file. The required data field is an unsigned longword that specifies the number of blocks in the file.

SMO$K_FLAGPAG
SMO$K_NOFLAGPAG

SMO$K_FLAGPAG specifies that a flag page be printed with a job. No data field is required.

SMO$K_NOFLAGPAG specifies that a flag page not be printed with a job. No data field is required.

SMO$K_FORMTYPE

SMO$K_FORMTYPE specifies the form number. The required 1-byte data field contains the form number.

SMO$K_GENDEV
SMO$K_NOGENDEV

SMO$K_GENDEV defines a queue as a generic queue. No data field is required.

SMO$K_NOGENDEV defines a queue as an execution queue. No data field is required.

SMO$K_GENPRT
SMO$K_NOGENPRT

SMO$K_GENPRT allows processing of jobs entered in a generic queue by this execution queue. No data field is required.

SMO$K_NOGENPRT disallows processing of jobs entered in a generic queue by this execution queue. No data field is required.

SMO$K_HOLD

SMO$K_HOLD specifies that a job be held until it is explicitly released. No data field is required.

SMO$K_INIPRI

SMO$K_INIPRI specifies the base priority of a batch job. The required 1-byte data field contains a priority value from 0 to 15.

SMO$K_JOBCOPY

SMO$K_JOBCOPY specifies that a job be repeated. The required 1-byte data field contains a number specifying how many times the job is to be repeated.

SMO$K_JOBLIM

SMO$K_JOBLIM specifies the maximum number of jobs that can be executed simultaneously in a batch queue. The required 1-byte data field contains this number.

SMO$K_JOBNAM

SMO$K_JOBNAM specifies the job name. The required data field is a counted ASCII string from 1 to 39 bytes.

SMO$K_JOBPRI

SMO$K_JOBPRI specifies priority for the queueing of a job. The required 1-byte data field contains a priority value from 0 through 255.

SMO$K_KEEPLOG
SMO$K_NOKEEPLOG

SMO$K_KEEPLOG specifies that the log file not be deleted after printing a batch job. No data field is required.

SMO$K_NOKEEPLOG specifies that the log file be deleted after printing a batch job. No data field is required.

SMO$K_LOGFNAM

SMO$K_LOGFNAM specifies the name of a log file for a job. The required data field contains a counted string specifying the name.

SMO$K_LOGQNAM

SMO$K_LOGQNAM specifies the name of a queue to which a batch job log file is to be spooled. The required data field contains a counted string specifying the name.

SMO$K_LOWER
SMO$K_NOLOWER

SMO$K_LOWER specifies that a printer must be equipped with lowercase characters. No data field is required.

SMO$K_NOLOWER specifies that a printer need not be equipped with lowercase characters. No data field is required.

SMO$K_MCPULM
SMO$K_NOMCPULM

SMO$K_MCPULM specifies the maximum CPU time for batch jobs. The required data field is an unsigned longword containing the desired number of 10 millisecond units of CPU time.

SMO$K_NOMCPULM specifies that no maximum CPU time is to be applied to batch jobs. No data field is required.

SMO$K_NEXTJOB

SMO$K_NEXTJOB terminates the current job and starts printing the next job in the queue. No data field is required.

SMO$K_NOFEED

SMO$K_NOFEED cancels automatic form feed for print jobs. No data field is required.

SMO$K_NOLOG

SMO$K_NOLOG specifies that no log file be kept for a batch job. No data field is required.

SMO$K_NOSPOOL

SMO$K_NOSPOOL specifies that a batch job log file not be spooled when the batch job completes. No data field is required.

SMO$K_NOTIFY
SMO$K_NONOTIFY

SMO$K_NOTIFY specifies that the user be notified (via BROADCAST) when a job has completed. No data field is required.

SMO$K_NONOTIFY specifies that the user is not to be notified when a job has completed. No data field is required.

SMO$K_PAGCNT

SMO$K_PAGCNT specifies the number of pages to print. The required 2-byte data field contains the number.

SMO$K_PAGHDR

SMO$K_PAGHDR specifies that a page heading be printed on the top of each output page. No data field is required.

SMO$K_PARAMS

SMO$K_PARAMS specifies parameters for a batch job. The required data field consists of one or more counted ASCII strings, terminated by 0.

SMO$K_REQDEST

SMO$K_REQDEST specifies the name of a queue in which to place jobs that have been requeued using SMO$K_REQUEUE. The required data field is a counted ASCII string.

SMO$K_REQPRIO

SMO$K_REQPRIO specifies a new priority for a job when the job is requeued using SMO$K_REQUEUE. The required data field is a priority value from 0 to 255.

SMO$K_REQUEUE

SMO$K_REQUEUE places an aborted print job back in the queue. No data field is required.

SMO$K_RLSTIM

SMO$K_RLSTIM specifies a time at which to release a held job. The required data field is a quadword containing a binary time value.

SMO$K_SMBNAME

SMO$K_SMBNAME specifies the name of a print symbiont for jobs originating from this queue. The required data field is a counted ASCII string containing the file name of the symbiont image.

SMO$K_SPCCNT

SMO$K_SPCCNT restarts the current job at some number of pages either previous to or subsequent to the page at which the job stopped. The required data field is a signed word containing the plus or minus page count.

SMO$K_TOPOFILE

SMO$K_TOPOFILE restarts the current job at the top of the file. No data field is required.

SMO$K_TRMDEV
SMO$K_NOTRMDEV

SMO$K_TRMDEV specifies that a generic queue can place jobs in terminal queues. No data field is required.

SMO$K_NOTRMDEV specifies that a generic queue can place jobs in printer queues. No data field is required.

SMO$K_WSDEFLT
SMO$K_NOWSDFT

SMO$K_WSDEFLT specifies the default working set size for a batch job. The default working set size must be less than or equal to the working set quota (SMO$K_WSQUOTA). The required data field is an unsigned word containing the number of pages.

SMO$K_NOWSDFT specifies that no working set default size be applied to this job. No data field is required.

SMO$K_WSDFLT
SMO$K_NOWSDFLT

SMO$K_WSDFLT specifies the default working set size for jobs originating from this queue. The default working set size must be less than or equal to the working set quota (SMO$K_WSQUTA). The required data field is an unsigned word containing the number of pages.

SMO$K_NOWSDFLT specifies that no default working set size is specified for jobs originating from this queue. No data field is required.

SMO$K_WSEXTNT
SMO$K_NOWSEXTNT

SMO$K_WSEXTNT specifies the working set extent for this batch job. The required data field is an unsigned word containing the number of pages.

SMO$K_NOWSEXTNT specifies that no working set extent is specified for this batch job. No data field is required.

SMO$K_WSQUOTA
SMO$K_NOWSQUO

SMO$K_WSQUOTA specifies the working set quota for this batch job. The required data field is an unsigned word containing the number of pages.

SMO$K_NOWSQUO specifies that no working set quota be applied to this job. No data field is required.

SMO$K_WSQUTA
SMO$K_NOWSQUTA

SMO$K_WSQUTA specifies the working set quota for jobs originating from this batch queue. The required data field is an unsigned word containing the number of pages.

SMO$K_NOWSQUTA specifies that no working set quota is specified for jobs originating from this batch queue. No data field is required.

SMO$K_WSXTANT
SMO$K_NOWSXTNT

SMO$K_WSXTANT specifies the default working set extent for jobs originating from this batch queue. The required data field is an unsigned word containing the number of pages.

SMO$K_NOWSXTNT specifies that no working set extent be applied to jobs originating from this batch queue. No data field is required.

The working set default size, the working set quota and the working set extent (maximum size) are included in each user record in the system user authorization file (UAF), and can be specified for individual jobs and/or for all jobs in a given queue.

A CPU time limit for the process is included in each user record in the system user authorization file (UAF). You can also specify any or all of the following: a CPU time limit for individual jobs, a default CPU time limit for all jobs in a given queue, and a maximum CPU time limit for all jobs in a given queue.

CONDITION VALUES RETURNED

SS$_NORMAL	Service successfully completed.
SS$_ACCVIO	The message buffer or buffer descriptor cannot be read by the caller.
SS$_BADPARAM	The specified message has a length of 0 or has more than 1000 characters.
SS$_DEVNOTMBX	The specified channel is not assigned to a mailbox.
SS$_INSFMEM	Insufficient system dynamic memory is available to complete the service, and the process has disabled resource wait mode with the Set Resource Wait Mode ($SETRWM) service.
SS$_IVCHAN	An invalid channel number was specified; that is, a channel number of 0 or a number larger than the number of channels available.
SS$_NOPRIV	The caller does not have write access to the specified mailbox.

CONDITION VALUES RETURNED IN THE MAILBOX

JBC$_NORMAL	Normal successful completion.
JBC$_DELACCESS	The file protection of the specified file, which was entered with the delete option, does not allow delete access to the caller.
JBC$_EMPTYJOB	The open job cannot be closed because it contains no files.
JBC$_EXECUTING	The parameters of the specified job cannot be modified because the job is currently executing.
JBC$_INCDSTQUE	The type of the specified destination queue is inconsistent with the requested operation.
JBC$_INCOMPLETE	The requested queue management operation cannot be executed because a previously requested queue management operation has not yet completed.
JBC$_INCQUETYP	The type of the specified queue is inconsistent with the requested operation.
JBC$_INVDSTQUE	The destination queue name is not syntactically valid.
JBC$_INVFUNCOD	The $SNDSMB request type is invalid.
JBC$_INVITMCOD	An optional item is invalid for the specified request type.
JBC$_INVMSGBUF	The message buffer is invalid because either it is not long enough to contain all required fields or an option extends beyond the end of the buffer.

JBC$_INVPARLEN	The length of a string specified with a $SNDSMB option is outside the valid range for that option.
JBC$_INVPARVAL	A specified option is outside the valid range.
JBC$_INVQUENAM	The queue name is not syntactically valid.
JBC$_JOBQUEDIS	The request cannot be executed because the system job queue manager has not been started.
JBC$_NODSTQUE	The specified destination queue does not exist.
JBC$_NOOPENJOB	The requesting process did not open a job with the SJC$_CREATE_JOB function.
JBC$_NOPRIV	The queue protection denies access to the queue for the specified operation.
JBC$_NOQUESPACE	The system job queue file was full and could not be extended.
JBC$_NORESTART	The specified job cannot be requeued because it was not defined to be restartable.
JBC$_NOSUCHFORM	The specified form does not exist.
JBC$_NOSUCHJOB	The specified job does not exist.
JBC$_NOSUCHQUE	The specified queue does not exist.
JBC$_NOTASSIGN	The specified queue cannot be deassigned because it is not assigned.
JBC$_REFERENCED	The specified queue cannot be deleted because of existing references by other queues or jobs.
JBC$_STARTED	The specified queue cannot be started because it is already running.

EXAMPLE

```
        $SMRDEF
REQUEST:                                ; start of message buffer
        .WORD   SMR$K_ALTER             ; alter queue
QUEUE:  .ASCIC  /SYS$PRINT______/       ; name of queue padded
                                        ; with spaces
RJOBID: .WORD   0                       ; destination of job identifier
OPT:    .BYTE   SMO$K_JOBPRI            ; alter the priority
NEWPRI: .BYTE   4                       ; destination of priority
                                        ; (default  value of 4)
        .BYTE   0                       ; end of message buffer
          .
          .
          .
        MOVW    JOBID,RJOBID            ; move job identifier to buffer
        MOVB    R6,NEWPRI               ; R6 has new priority
        $SNDSMB_S -
                MSGBUF=REQUEST, -
                CHAN=MBXCHAN
```

The above example shows a segment of a program used to alter the priority of a job in the queue SYS$PRINT.

$TRNLOG Translate Logical Name

The Translate Logical Name service searches the logical name tables for a specified logical name and returns an equivalence name string into a user-specified buffer; the search is not iterative. The process, group, and system logical name tables are searched in that order.

The Translate Logical Name ($TRNLNM) service supersedes the $TRNLOG service. New programs should be written using $TRNLNM, not $TRNLOG, and old programs that use $TRNLOG should be converted to use $TRNLNM.

FORMAT

SYS$TRNLOG *lognam ,[rsllen] ,rslbuf ,[table] ,[acmode] ,[dsbmsk]*

RETURNS

VMS usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

Longword condition value. All system services return (by value) a condition value in R0. Condition values that can be returned by this service are listed under "CONDITION VALUES RETURNED."

ARGUMENTS

lognam

VMS usage: **logical_name**
type: **character-coded text string**
access: **read only**
mechanism: **by descriptor–fixed length string descriptor**

Name of the logical name to be translated. The **lognam** is the address of a character string descriptor pointing to the logical name string.

rsllen

VMS usage: **word_unsigned**
type: **word (unsigned)**
access: **write only**
mechanism: **by reference**

Length in bytes of the equivalence name to which the logical name translates. The **rsllen** argument is the address of a word to receive this length.

rslbuf

VMS usage: **char_string**
type: **character-coded text string**
access: **write only**
mechanism: **by descriptor–fixed length string descriptor**

Buffer into which $TRNLOG writes the equivalence name. The **rslbuf** argument is the address of a character string descriptor pointing to this buffer.

table

VMS usage: **byte_unsigned**
type: **byte (unsigned)**
access: **write only**
mechanism: **by reference**

Logical name table in which the equivalence name was found. The **table** argument is the address of a byte into which $TRNLOG writes a value. The value 0 specifies the system logical name table; 1, the group table; and 2, the process table.

acmode

VMS usage: **access_mode**
type: **byte (unsigned)**
access: **write only**
mechanism: **by reference**

Access mode associated with the logical name and equivalence name. The **acmode** argument is the address of a byte into which $TRNLOG writes this access mode. The $PSLDEF macro defines the symbols for the four access modes. The contents of this byte is valid only if the equivalence name was found in the process logical name table (table 2).

dsbmsk

VMS usage: **mask_longword**
type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

Logical name tables that are not to be searched by $TRNLOG. The **dsbmsk** argument is a longword bit vector wherein a bit, when set, disables the search of the corresponding logical name table. Bit 0 corresponds to the system logical name table; bit 1, to the group logical name table; and bit 2, to the process logical name table.

If **dsbmsk** is not specified or is specified as 0 (the default), all three logical name tables are searched.

DESCRIPTION

If the first character of a specified logical name is an underscore character (_), no translation is performed. However, the underscore character is removed from the string and the modified string is returned in the output buffer.

CONDITION VALUES RETURNED

SS$_NORMAL	Service successfully completed. An equivalence name was returned.
SS$_NOTRAN	Service successfully completed. An equivalence name was not found; the logical name was returned in the buffer.
SS$_ACCVIO	The logical name string or string descriptor cannot be read by the caller; or the output length, output buffer, or table or access mode field cannot be written by the caller.
SS$_IVLOGNAM	The specified logical name string has a length of 0 or has more than 255 characters.
SS$_RESULTOVF	The buffer was too small to receive the returned string.

3 Obsolete RTL Routines

This chapter describes obsolete RTL routines. An *obsolete routine* is a routine that has been superseded by a more efficient or more flexible routine. Obsolete routines are no longer updated.

Section 3.1 through Section 3.3 describe RTL routines by functional groups. Section 3.4 contains a complete description of each obsolete RTL routine.

DIGITAL recommends that you use current routines when you write new programs. Old programs which use obsolete routines should be updated to make future maintenance easier.

3.1 Obsolete FOR$ Routines

Table 3–1 displays the obsolete FOR$ routines described in this chapter and the current OTS$ routines that have replaced them.

Table 3–1 Table of Obsolete RTL Routines

Obsolete Routine	New Routine
FOR$CNV_OUT_I	OTS$CVT_L_TI
FOR$CNV_OUT_L	OTS$CVT_L_TL
FOR$CNV_OUT_O	OTS$CVT_L_TO
FOR$CNV_OUT_Z	OTS$CVT_L_TZ
FOR$CNV_IN_I	OTS$CVT_TI_L
FOR$CNV_IN_L	OTS$CVT_TL_L
FOR$CNV_IN_O	OTS$CVT_TO_L
FOR$CNV_IN_DEFG	OTS$CVT_T_Z
FOR$CNV_IN_Z	OTS$CVT_TZ_L

The obsolete FOR$ routines listed in the first column of Table 3–1 were the original routines. FOR$ routines were written to support conversion and I/O functions in FORTRAN. As more languages evolved that also needed these types of services, the FOR$ routines were rewritten as OTS$ routines. OTS$ routines are generic routines and are language independent. Therefore, a BASIC compiler can call an OTS$ routine, but may not call the equivalent FOR$ routine. It is interesting to note that even the FORTRAN compiler now uses the OTS$ entry points rather than the FOR$ entry points.

The FOR$ and OTS$ routines listed above are equivalent. In fact, each pair not only performs the same function, but does so using the same code. Each routine module contains two entry points, one for OTS$ and one for FOR$. These two entry points lead to a common code path.

Obsolete RTL Routines

3.1 Obsolete FOR$ Routines

In some of the routines, however, there is a slight difference between OTS$ and FOR$. Many of the optional arguments provided in the OTS$ routines are not provided in their FOR$ equivalents. In each case where the FOR$ routine does not include optional arguments present in the OTS$ version, the default values of the optional arguments are used.

Because OTS$ now provides the same services as FOR$, with the added advantage of being language independent, DIGITAL strongly recommends that you use only OTS$ routines.

3.2 Obsolete Terminal-Independent Screen Manipulation Procedures

These obsolete screen manipulation procedures have been superseded by the SMG$ routines. DIGITAL recommends that you use only SMG$ routines for screen manipulation.

The obsolete terminal-independent screen procedures were designed to allow high-level language programs to manipulate output to terminals and files. Table 3–2 lists these obsolete screen procedures and their functions.

When you write code that uses these procedures, you do not need to know what kind of terminal will be used when your program runs. They are intended primarily for controlling output to video terminals, but they also function predictably if the output device is a hardcopy terminal or a file. The word *screen* refers to the screen of a video terminal.

Each of these procedures also has a corresponding SCR$ entry point, where scalar input arguments are passed by immediate value. For example, if you use SCR$SET_CURSOR rather than LIB$SET_CURSOR, you must pass the line number and column number by immediate value.

Note: These procedures set the terminal type to correspond to the type specified by the /DEVICE_TYPE qualifier of the SET TERMINAL command.

Table 3–2 The Terminal-Independent Screen Procedures

Entry Point	Function
LIB$SCREEN_INFO SCR$SCREEN_INFO	Returns terminal specifications to your program
LIB$DOWN_SCROLL SCR$DOWN_SCROLL	Moves the cursor up one line, or scrolls down if at top
LIB$ERASE_LINE SCR$ERASE_LINE	Erases all of the character positions on the screen from the specified cursor position to the end of the line
LIB$ERASE_PAGE SCR$ERASE_PAGE	Erases all of the character positions on the screen from the specified cursor position to the end of the screen
LIB$SET_CURSOR SCR$SET_CURSOR	Sets the cursor to the specified position on the screen
LIB$SET_SCROLL SCR$SET_SCROLL	Establishes a scrolling region
LIB$UP_SCROLL SCR$UP_SCROLL	Moves the cursor down one line, or scrolls up if at bottom
LIB$GET_SCREEN SCR$GET_SCREEN	Accepts input from a terminal and puts it into a user-specified buffer
LIB$PUT_LINE SCR$PUT_LINE	Displays the specified text at the current cursor position followed by a specified number of line advances
LIB$PUT_SCREEN SCR$PUT_SCREEN	Puts specified text to the screen beginning at a specified line and column
LIB$SET_OUTPUT SCR$SET_OUTPUT	Creates a channel to the specified terminal or buffer
LIB$STOP_OUTPUT SCR$STOP_OUTPUT	Closes the channel to the specified terminal or buffer
LIB$PUT_BUFFER SCR$PUT_BUFFER	Puts the current buffer to the screen or to the previous buffer
LIB$SET_BUFFER SCR$SET_BUFFER	Sets or clears buffer mode

3.2.1 Obtaining Screen Information

LIB$SCREEN_INFO determines information about the terminal currently being used and places it in a storage area that you provide. The following information is available:

- Special terminal characteristics
  - DIGITAL video terminal
  - Hardcopy terminal, unknown terminal type, or file
  - ANSI terminal (VT100-compatible)
  - The terminal has Advance Video Option
  - The terminal understands REGIS graphics
  - The terminal is a block mode terminal

- Type of terminal
  - Unknown type (=SET TERMINAL/DEVICE_TYPE=UNKNOWN)
  - VT52 (=SET TERMINAL/DEVICE_TYPE=VT52)
  - VT100 (=SET TERMINAL/DEVICE_TYPE=VT100)
  - Or any of the other VMS terminal types
- Line width
- Lines per page

3.2.2 Positioning the Cursor on the Screen

Screen procedures let you customize interactive input and output by manipulating the cursor, erasing parts of the screen display, and setting the scrolling region. You can also cause the screen to scroll up or down a specified number of lines. Furthermore, you can write programs that perform many of these functions without knowing what type of DIGITAL terminal will be used. If the terminal is unable to perform a particular function, it will usually ignore the procedure call and continue, without issuing an error message.

These procedures use the following conventions:

- The top line of a screen is line number one.
- The leftmost column of a screen is column number one.
- When the line and column arguments are optional, you must specify both agruments or neither.

These procedures are designed to display the information you have passed to them as accurately as possible. If you commit a formatting error, such as entering too much text for the display area or moving the cursor to an invalid position, you do not want your program to exit. Therefore, the screen procedures ignore such errors rather than treat them as fatal. For example:

- The screen procedures do not check for cursor position specifications that exceed the maximum number of lines or columns for the terminal.
- The screen procedures do not insert carriage returns in order to cause lines to wrap.
- The screen procedures do not try to prevent the loss of text characters, when text is positioned beyond the screen boundaries.

3.2.2.1 Controlling Input from and Output to the Screen

Three procedures let your program control I/O to and from the screen.

- LIB$GET_SCREEN reads an input string from SYS$INPUT into a destination text string. The destination string can be fixed-length or dynamic. You can supply two optional arguments to this procedure:

prompt-str	Specifies text string to be displayed on the terminal before the procedure accepts the contents of the screen as input.
out-len	Will contain the actual number of characters written into the destination string.

- LIB$PUT_SCREEN displays the contents of the specified text string on the screen. This procedure accepts three optional arguments:

line-no, col-no	The line number and column number at which the displayed text will begin. If you specify one of these arguments, you must specify both.
flags	A longword value that specifies the special graphics attributes, such as blinking and reverse video, available on some terminals. Currently, only the first four bits of the longword are used. Table 3–3 indicates the binary value corresponding to each combination of screen attributes currently available.

- LIB$PUT_LINE displays a single line of text on the screen and then moves the cursor. Its arguments specify the address of the text and the number of lines to move the cursor. The second argument is a signed value. If it is negative, the cursor moves upward after the line is displayed.

  By default, the cursor moves to the beginning of the next line. The effect is the same as a carriage return and line feed.

Table 3–3 Screen Attributes

Binary value	Decimal value	Attributes
0000	0	None
0001	1	Bold
0010	2	Reverse Video
0011	3	Bold, Reverse Video
0100	4	Blinking
0101	5	Blinking, Bolded
0110	6	Blinking, Reverse Video
0111	7	Blinking, Reverse Video, Bolded
1000	8	Underlined
1001	9	Underlined, Bolded
1010	10	Underlined, Reverse Video
1011	11	Underlined, Reverse Video, Bolded
1100	12	Underlined, Blinking
1101	13	Underlined, Blinking, Bolded
1110	14	Underlined, Blinking, Reverse Video
1111	15	Underlined, Blinking, Reverse Video, Bolded

LIB$SET_OUTPUT allows you to direct output to a terminal or file other than the default output device, SYS$OUTPUT. This procedure can be used to allow a single process to display information on multiple terminals. To do this, your program calls LIB$SET_OUTPUT once for each terminal, before sending output. You can also use it to direct output to a file.

The first time you call this procedure, you must use all the arguments in order to set up the channel to the device or file. After this, you can simply use the file specification or device name for each call and user routines and user arguments, if any.

If you call LIB$SET_OUTPUT with no arguments, the output goes to SYS$OUTPUT, the default process output stream.

LIB$STOP_OUTPUT can be used to close the output stream established by LIB$SET_OUTPUT. If you do not call LIB$STOP_OUTPUT, the channel will be freed automatically when the image exits.

LIB$STOP_OUTPUT is useful, for example, if you wish to make multiple versions of a single file. If you call LIB$SET_OUTPUT, several times to send output to a file and do not call LIB$STOP_OUTPUT, the original file will be overwritten. All of the output except the last version will be lost. If you call LIB$STOP_OUTPUT after each call to LIB$SET_OUTPUT, however, LIB$SET_OUTPUT will open a new version of the file each time, and all of the output will be preserved.

3.2.2.2 Buffering Screen I/O

Normally, when your program displays text on the terminal, the Queue I/O ($QIO) system service is called at least once for each call to a screen procedure. However, buffer mode may also be used with the screen procedures to format output to a hardcopy file or terminal.

The use of buffer mode involves four steps:

1. You set up an area of storage to act as the buffer.
2. You establish buffer mode in your program by calling LIB$SET_BUFFER.
3. The program calls other screen procedures, such as LIB$PUT_SCREEN. Instead of displaying text on the screen, these procedures put the text into the buffer.
4. The program writes the contents of the buffer on the screen by calling LIB$PUT_BUFFER. This is normally done when enough information has accumulated to fill the screen.

The amount of data that fills the screen cannot always be determined ahead of time. For this reason, when the data overflows the buffer, $QIO is called automatically to display the buffer's contents. Then LIB$SET_BUFFER sets the buffer data size to zero and continues the current buffer mode by placing new data in the current buffer.

Modular programs can use screen buffering at several levels. That is, a procedure can establish buffer mode, then call another procedure which also establishes buffer mode, and so on.

Each procedure that calls LIB$SET_BUFFER to establish buffer mode must also set aside storage for a buffer. However, only one buffer is active at a time. When a called procedure establishes a buffer, LIB$SET_BUFFER copies the contents of the previously established buffer into the current one, and sets the previous buffer to "empty."

A pointer to the buffer established previously is available to the called procedure. You can access this pointer by declaring a longword to contain it and by passing it as an argument to LIB$SET_BUFFER or LIB$PUT_BUFFER. If you call LIB$SET_BUFFER using the address of the previous buffer as an argument, this address is saved. Then you can use it as an argument for LIB$PUT_BUFFER, to copy the current buffer back to the calling program's buffer before returning to the calling program.

For example, assume that you want to write a procedure A that calls a procedure B, and that you will use a buffer to accumulate the output from both A and B. Procedure A performs the following:

1. Calls LIB$SET_BUFFER to establish a buffer called ABUF.
2. Puts text into ABUF.
3. Calls procedure B.

Procedure B performs the following:

1. Declares a longword to save the address of ABUF.
2. Sets up a buffer for itself, called BBUF.

3 Calls LIB$SET_BUFFER, using as arguments BBUF and the address of ABUF. This establishes buffer mode for procedure B, specifies the new buffer, and saves the address of ABUF.

4 Places into BBUF its own additions to the text.

5 Calls LIB$PUT_BUFFER, using the longword (the address of ABUF) as the argument. This copies BBUF contents to ABUF.

This call passes the contents of BBUF back to the procedure A's buffer. At this point, procedure B returns control to procedure A. Its buffer now contains all the text generated by both procedures. It then can display the text or pass it up to a higher level by calling LIB$PUT_BUFFER.

Because of this process of copying the buffer's contents from one procedure to the next, the contents of the buffer accumulate from the time that the buffer mode is first established. This means that when you set up the buffer for the main procedure, you must make it large enough to contain all the information created by all of the procedures it calls, not just the output from the called procedure itself. Otherwise, the data will overflow the buffer, and an automatic $QIO will occur. Similarly, the calling procedure must set up a buffer large enough to contain the data that will be buffered by all the procedures called at lower levels, in addition to its own output.

LIB$SET_BUFFER takes two arguments. The first (required) argument specifies the buffer being established. The second (optional) argument saves the address of the previously established buffer. In order to preserve modularity, the main program in this example uses both of these arguments. Using LIB$SET_BUFFER and LIB$PUT_BUFFER in pairs also preserves modularity by ensuring that the contents of the buffer are predictable at any point in the execution of the program.

Calling LIB$PUT_BUFFER with no argument causes the contents of the buffer to be flushed to the screen. If LIB$PUT_BUFFER is called with the **buffer** argument set to zero, buffer mode stops automatically but the current contents of the buffer are lost. You should call only these procedures with **buffer** equal to zero if a situation occurs (such as a call to LIB$STOP or SYS$EXIT) that will prevent your procedure from returning to its caller and thus printing the contents of the buffer.

3.2.2.3 Using Screen Procedures with Files and Hardcopy Terminals

The terminal-independent screen procedures will execute if the output device is a hardcopy terminal or a file instead of a video terminal. The output of a program on these devices, however, may not be precisely comparable to the same program's output on a screen. When the screen procedures are used to output text to something other than a video screen, they use VMS Record Management Services (RMS) to format the output, rather than the $QIO system service. Unlike $QIO, RMS adds a carriage return and line feed to each line.

As with video terminals, you can use the screen procedures to write to a hardcopy terminal or file either with or without buffering. If you use the procedures without establishing buffer mode, all features relating to the formatting of the screen are ignored. The procedures will ignore arguments specifying cursor position, and all output will be done line by line. For example, LIB$UP_SCROLL and LIB$DOWN_SCROLL will do nothing, and text displayed by LIB$PUT_SCREEN and LIB$PUT_LINE will be followed by a carriage return/line feed.

However, if you use the procedures after establishing buffer mode, the output in the file or on the hardcopy page will look as much as possible like the corresponding screen display. When you call LIB$SET_BUFFER and specify a hardcopy device or a file, the procedure establishes its own buffer, equivalent to a logical "screen." It then moves the cursor and places text within that buffer as though the buffer were the screen. The arguments of LIB$PUT_SCREEN that specify bolding and underlining also cause bolding and underlining on hardcopy terminals and in files; the blinking and reverse video options are ignored.

When your program calls LIB$PUT_BUFFER to display the contents of the buffer, the procedure scans the buffer from top to bottom, placing each line in the file or on the hardcopy page and adding a carriage return and line feed. If the procedure finds more than one blank line at the bottom, it stops the output and issues a form feed.

Errors that occur during output to hardcopy terminals and files are handled like other screen formatting errors: the procedures do the best they can to reproduce the screen display without causing the image to exit. For example, overflow does not cause the buffer's contents to be output. Rather, the buffer is displayed only by a call to LIB$PUT_BUFFER; any text that exceeded the limits of the buffer is lost. Transfer of information beyond the end of a line causes line wrap; any text that already exists on the next line is overwritten and lost. If you move the cursor to a point within the buffer and write text that will overflow, or if you move the cursor to a point beyond the limits of the buffer, no error message results. The procedure does the best it can, and the excess text is lost. Because of this "best try" error handling, you should structure your output carefully when writing output to hardcopy terminals and files to avoid overflow and line wrap.

3.3 LIB$EMULATE

The functions that were previously performed by LIB$EMULATE are now done automatically. Although no error will result when you call this routine, it is no longer necessary to do so.

3.4 Obsolete RTL Routines

This reference section contains obsolete RTL routines.

FOR$CNV_OUT_I Convert Signed Integer to Decimal Text

FOR$CNV_OUT_I converts a signed integer to a decimal ASCII text string. This procedure supports FORTRAN Iw and Iw.m output.

FORMAT

FOR$CNV_OUT_I *value,out-str*

RETURNS

type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

value

type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Signed integer value that FOR$CNV_OUT_I converts to a decimal ASCII text string. The **value** argument is the address of this integer value.

out-str

type: **character string**
access: **write only**
mechanism: **by descriptor, fixed-length**

Decimal ASCII text string that FOR$CNV_OUT_I creates when it converts the signed integer to a decimal ASCII text string. The **out-str** argument is the address of a descriptor pointing to this text string. The string is assumed to be fixed-length (DSC$K_CLASS_S).

CONDITION VALUES RETURNED

SS$_NORMAL	Routine successfully completed
OTS$_OUTCONERR	Output conversion error. The result would have exceeded the fixed-length string; the output string is filled with asterisks.

FOR$CNV_OUT_L Convert Integer to Logical Text

FOR$CNV_OUT_L converts an integer to the ASCII text string representation using FORTRAN L (logical) format.

FORMAT

FOR$CNV_OUT_L *value, out-str*

RETURNS

type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

value

type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Signed integer value that FOR$CNV_OUT_L converts to an ASCII text string. The **value** argument is the address of this integer value.

out-str

type: **character string**
access: **write only**
mechanism: **by descriptor, fixed-length**

Output string that FOR$CNV_OUT_L creates when it converts the integer value to an ASCII text string. The **out-str** argument is the address of a descriptor pointing to this ASCII text string.

The output string is assumed to be fixed-length (DSC$K_CLASS_S).

The output string consists of *(length-1)* blanks followed by:

The letter T if bit 0 is set
The letter F if bit 0 is clear

CONDITION VALUES RETURNED

SS$_NORMAL	Routine successfully completed.
OTS$_OUTCONERR	Output conversion error. The result would have exceeded the fixed-length string; the output string is of zero length (DSC$W_LENGTH=0).

FOR$CNV_OUT_O Convert Unsigned Integer to Octal Text

FOR$CNV_OUT_O converts an unsigned integer to an octal ASCII text string. FOR$CNV_OUT_O supports FORTRAN Ow and Ow.m output conversion formats.

FORMAT

FOR$CNV_OUT_O *value, out-str*

RETURNS

type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

value

type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Integer value that FOR$CNV_OUT_O converts to an octal ASCII text string. The **value** argument is the address of this integer value.

out-str

type: **character string**
access: **write only**
mechanism: **by descriptor, fixed-length**

Output string that FOR$CNV_OUT_O creates when it converts the integer value to an octal ASCII text string. The **out-str** argument is the address of a descriptor pointing to the octal ASCII text string. The string is assumed to be fixed-length (DSC$K_CLASS_S).

CONDITION VALUES RETURNED

SS$_NORMAL	Routine successfully completed.
OTS$_OUTCONERR	Output conversion error. The result would have exceeded the fixed-length string; the output string is filled with asterisks.

FOR$CNV_OUT_Z Convert Integer to Hexadecimal Text

FOR$CNV_OUT_Z converts an unsigned integer to a hexadecimal ASCII text string. FOR$CNV_OUT_Z supports FORTRAN Zw and Zw.m output conversion formats.

FORMAT

FOR$CNV_OUT_Z *value, out-str*

RETURNS

type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

value

type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Integer value that FOR$CNV_OUT_Z converts to a hexadecimal ASCII text string. The **value** argument is the address of this integer value.

out-str

type: **character string**
access: **write only**
mechanism: **by descriptor, fixed-length**

Output string that FOR$CNV_OUT_Z creates when it converts the integer value to a hexadecimal ASCII text string. The **out-str** argument is the address of a descriptor pointing to this ASCII text string. The string is assumed to be fixed-length (DSC$K_CLASS_S).

CONDITION VALUES RETURNED

SS$_NORMAL	Routine successfully completed.
OTS$_OUTCONERR	Output conversion error. The result would have exceeded the fixed-length string; the output string is filled with asterisks.

FOR$CNV_IN_I Convert Signed Integer Text to Integer

FOR$CNV_IN_I converts an ASCII text string representation of a decimal number to a signed byte, word, or longword integer value.

FORMAT

FOR$CNV_IN_I *inp-str, value[,value-size][,flags]*

RETURNS

type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

inp-str

type: **character string**
access: **read only**
mechanism: **by descriptor, fixed-length**

Input ASCII text string that FOR$CNV_IN_I converts to a signed byte, word, or longword. The **imp-str** argument is the address of a descriptor pointing to the input string.

The syntax of a valid ASCII text input string is:

```
[+ or -][<integer-digits>]
```

FOR$CNV_IN_I always ignores leading blanks. A decimal point is assumed at the right of the input string.

value

type: **unspecified**
access: **write only**
mechanism: **by reference**

Signed byte, word, or longword integer value (depending on **value-size**) that FOR$CNV_IN_I creates when it converts the ASCII text string. The **value** argument is the address of the integer value.

value-size

type: **longword integer (signed)**
access: **read only**
mechanism: **by value**

Number of bytes occupied by the value that FOR$CNV_IN_I creates when it converts the ASCII text string to an integer value. Valid values for the **value-size** argument are one, two, and four. The contents of **value-size** determine whether the integer value that FOR$CNV_IN_I creates is a byte, word, or longword. If an invalid value is given, FOR$CNV_IN_I returns an error. This is an optional argument. If omitted, the default is four and FOR$CNV_IN_I returns a longword integer.

flags

type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

User-supplied flags which FOR$CNV_IN_I uses to determine how blanks and tabs are interpreted.

Bit	Description
0	If set, FOR$CNV_IN_I ignores all blanks. If clear, FOR$CNV_IN_I ignores leading blanks, but interprets blanks after the first legal character as zeros.
4	If set, FOR$CNV_IN_I ignores tabs. If clear, FOR$CNV_IN_I interprets tabs as invalid characters.

This is an optional argument. If omitted, the default is that all bits are set and FOR$CNV_IN_I ignores blanks and tabs.

CONDITION VALUES RETURNED

SS$_NORMAL	Routine successfully completed.
OTS$_INPCONERR	Input conversion error; an invalid character in the input string, or the value overflows byte, word, or longword, or **value-size** is invalid; **value** is set to zero.

FOR$CNV_IN_L Convert Logical Text to Integer

FOR$CNV_IN_L converts an ASCII text string representation of a FORTRAN–77 L format to a byte, word, or longword integer value. The result is a longword by default, but the calling program can specify a byte or a word value instead.

FORMAT

FOR$CNV_IN_L *inp-str ,value[,value-size]*

RETURNS

type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

inp-str

type: **character string**
access: **read only**
mechanism: **by descriptor, fixed-length**

Input string containing an ASCII text representation of a FORTRAN–77 L format that FOR$CNV_IN_L converts to a byte, word, or longword integer value. The **inp-str** argument is the address of a descriptor pointing to the input string.

The syntax of a valid ASCII text input string is:

```
        <zero or more blanks>
        <     <end of string>
                      or
              <       <"." or nothing>
Letter:                       <"T", "t", "F", or "f">
                       <zero or more of any character>
                       <end of string>>>
```

value

type: **unspecified**
access: **write only**
mechanism: **by reference**

Integer value that FOR$CNV_IN_L creates when it converts the ACSII text input string. The **value** argument is the address of this integer value. FOR$CNV_IN_L returns a minus one as the contents of the **value** argument if the character denoted by "Letter:" is "T" or "t." Otherwise, FOR$CNV_IN_L sets **value** to zero.

value-size

type: **longword integer (signed)**
access: **read only**
mechanism: **by value**

Number of bytes occupied by the integer value that FOR$CNV_IN_L creates when it converts the ASCII text input string. The **value-size** argument contains the number of bytes. Valid values are one, two, and four. These

values determine whether FOR$CNV_IN_L returns a byte, word, or longword integer value. If an invalid value is given, FOR$CNV_IN_L returns an error. This is an optional argument. If omitted, the default is four and FOR$CNV_IN_L returns a longword integer value.

CONDITION VALUES RETURNED

SS$NORMAL	Routine successfully completed.
OTS$_INPCONERR	Invalid character in the input string or invalid **value-size; value** is set to zero.

FOR$CNV_IN_O Convert Octal Text to Signed Integer

FOR$CNV_IN_O converts an ASCII text string representation of an unsigned octal value to an unsigned integer of an arbitrary length. The result is a longword by default, but the calling program can specify any number of bytes.

FORMAT

FOR$CNV_IN_O *inp-str ,value[,value-size][,flags]*

RETURNS

type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

inp-str

type: **character string**
access: **read only**
mechanism: **by descriptor, fixed-length**

Input string containing an ASCII text string representation of an unsigned octal value that FOR$CNV_IN_O converts to an unsigned integer. The **inp-str** argument is the address of a descriptor pointing to the input string. The valid input characters are the space and the digits 0 through 7. No sign is permitted.

value

type: **unspecified**
access: **write only**
mechanism: **by reference**

Integer value that FOR$CNV_IN_O creates when it converts the input string. The **value** argument is the address of the unsigned integer value.

value-size

type: **longword integer (signed)**
access: **read only**
mechanism: **by value**

Number of bytes occupied by the unsigned integer value. The **value-size** argument contains the number of bytes. If the content of the **value-size** argument is zero or a negative number, FOR$CNV_IN_O returns an error. This is an optional argument. If omitted, the default is four and FOR$CNV_IN_O returns a longword integer.

flags

type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

User-supplied flags that FOR$CNV_IN_O uses to determine how blanks within the input string are interpreted. The **flags** argument contains the user-supplied flags.

Bit 0 If set, FOR$CNV_IN_O ignores all blanks. If clear, FOR$CNV_IN_O interprets blanks as zeros.

This is an optional argument. If omitted, the default is that all bits are clear.

CONDITION VALUES RETURNED

SS$_NORMAL	Routine successfully completed.
OTS$_INPCONERR	Input conversion error. An invalid character, overflow, or invalid **value-size** occurred.

FOR$CNV_IN_DEFG Convert Numeric Text to Floating

FOR$CNV_IN_DEFG converts an ASCII text string representation of a numeric value to a D_floating value. The routine supports FORTRAN D, E, F, and G input type conversion as well as similar types for other languages.

FORMAT

FOR$CNV_IN_DEFG *inp-str ,value [,digits-in-fract] [,scale-factor] [,flags] [,extbits]*

RETURNS

type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

inp-str

type: **character string**
access: **read only**
mechanism: **by descriptor, fixed-length**

Input string containing an ASCII text string representation of a numeric value that FOR$CNV_IN_DEFG converts to a D_floating value. The **inp-str** argument is the address of a descriptor pointing to the address of the input string.

The syntax of a valid input string is:

```
<zero or more blanks>
<"+", "-", or nothing>
<zero or more decimal digits>
<"." or nothing>
<zero or more decimal digits>
<exponent or nothing, where exponent is:
        <    <<"E", "e". "D", "d", "Q", or "q">
                <zero or more blanks>
                <"+", "-", or nothing>>>
                or
                <"+", or "-">>
        <zero or more decimal digits>>
<end of string>
```

There is no difference in semantics between any of the six valid exponent letters (E, e, D, d, Q, q).

value

type: **D_floating**
access: **write only**
mechanism: **by reference**

Floating-point value that FOR$CNV_IN_DEFG creates when it converts the input string. The **value** argument is the address of a D_floating number containing the value.

digits-in-fract

type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

Number of digits in the fraction if no decimal point is included in the input string. The **digits-in-fract** argument contains the number of digits. This is an optional argument. If omitted, the default is zero.

scale-factor

type: **longword integer (signed)**
access: **read only**
mechanism: **by value**

Scale factor. The **scale-factor** argument contains the value of the scale factor. If flags bit 6 is clear, the resultant value is multiplied by 10**factor unless the exponent is present. If flags bit 6 is set, the scale factor is always applied. This is an optional argument. If omitted, the default is zero.

flags

type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

User-supplied flags.

Bit 0	If set, FOR$CNV_IN_DEFG ignores blanks. If clear, FOR$CNV_IN_DEFG interprets blanks as zeros.
Bit 1	If set, FOR$CNV_IN_DEFG allows only E or e exponents. If clear, FOR$CNV_IN_DEFG allows E, e, D, d, Q, and q exponents. (Bit 1 would be clear for BASIC and set for FORTRAN).
Bit 2	If set, FOR$CNV_IN_DEFG interprets an underflow as an error. If clear, FOR$CNV_IN_DEFG does not interpret an underflow as an error.
Bit 3	If set, FOR$CNV_IN_DEFG truncates the value. If clear, FOR$CNV_IN_DEFG rounds the value.
Bit 4	If set, FOR$CNV_IN_DEFG ignores tabs. If clear, FOR$CNV_IN_DEFGz interprets tabs as invalid characters.
Bit 5	If set, an exponent must begin with a valid exponent letter. If clear, the exponent letter may be omitted.
Bit 6	If set, FOR$CNV_IN_DEFG always applies the scale factor. If clear, FOR$CNV_IN_DEFG applies the scale factor only if there is no exponent present in the string.

If **flags** is omitted, all bits are clear.

ext-bits

type: **word integer (signed)**
access: **write only**
mechanism: **by reference**

The extra precision bits. If present, **value** is not rounded, and the first 8 bits after truncation are returned as a byte in this argument. This value is suitable for use as the extension operand in an EMOD instruction.

CONDITION VALUES RETURNED

SS$_NORMAL	Routine successfully completed.
OTS$_INPCONERR	Input conversion error; an invalid character in the input string, or the value is outside the range that can be represented. **Value** is set to +0.0 (not reserved operand -0.0).

FOR$CNV_IN_Z Convert Hexadecimal Text to Unsigned Integer

FOR$CNV_IN_Z converts an ASCII text string representation of an unsigned hexadecimal value to an unsigned integer of an arbitrary length. By default, the result is a longword, but the calling program can specify any number of bytes.

FORMAT

FOR$CNV_IN_Z *inp-str ,value[,value-size][,flags]*

RETURNS

type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

inp-str

type: **character string**
access: **read only**
mechanism: **by descriptor, fixed-length**

Input string containing an ASCII text string representation of an unsigned hexadecimal value the FOR$CNV_IN_Z converts to an unsigned integer. The **inp-str** argument is the address of a descriptor pointing to the input string. Valid input characters are the space, the digits 0 through 9, and letters A through F. No sign is permitted. Lowercase letters a through f are acceptable.

value

type: **unspecified**
access: **write only**
mechanism: **by reference**

Integer value created when FOR$CNV_IN_Z converts the input string. The **value** argument is the address of the integer value.

value-size

type: **longword integer (signed)**
access: **read only**
mechanism: **by value**

Number of bytes occupied by the integer value. The **value-size** argument contains the number of bytes. If the value size is zero or a negative number, FOR$CNV_IN_Z returns an input conversion error. This is an optional argument. If omitted, the default is four.

flags

type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

User-supplied flags that FOR$CNV_IN_Z uses to determine how blanks are interpreted.

Bit 0	If set, FOR$CNV_IN_Z ignores blanks. If set, FOR$CNV_IN_Z interprets blanks as zeros.

This is an optional argument. If omitted, the default is that all bits are clear.

CONDITION VALUES RETURNED

SS$_NORMAL	Routine successfully completed.
OTS$_INPCONERR	Input conversion error. An invalid character, overflow, or invalid **value-size** occurred.

LIB$DOWN_SCROLL Down Scroll the Screen (Move Cursor Up One Line)

LIB$DOWN_SCROLL moves the cursor up one line on the screen. If the cursor is already at the top line on the screen, all lines move down one line, the top line is replaced with a blank line, and the data that was on the bottom line is lost. If a scrolling region is active, then the above logic applies to the top and bottom lines of the scrolling region.

FORMAT

LIB$DOWN_SCROLL

RETURNS

type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

CONDITION VALUES RETURNED

SS$_NORMAL	Routine sucessfully completed.

EXAMPLE

```
CALL LIB$SET_CURSOR (1, 1)
CALL LIB$DOWN_SCROLL ()
```

This FORTRAN code fragment causes the text on the screen to be scrolled down one line.

LIB$EMULATE Emulate Execution of VAX Instructions

LIB$EMULATE is a condition handler that emulates excution of VAX instructions that are not implemented on the host processor. If LIB$EMULATE can emulate the instruction, the exception essentially disappears. The instructions emulated are the G_floating, H_floating, and octaword integer instructions.

FORMAT

LIB$EMULATE *sig-args ,mch-args*

RETURNS

type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

sig-args

type: **longword (unsigned)**
access: **modify**
mechanism: **by reference, array reference**

Signal argument vector. The **sig-args** argument is the address of an unsigned longword array containing the signal argument vector.

mch-args

type: **longword (unsigned)**
access: **modify**
mechanism: **by reference, array reference**

Mechanism argument vector. The **mch-args** argument is the address of an unsigned longword array containing the mechanism argument vector.

DESCRIPTION

The preferred use of LIB$EMULATE is to establish it as a condition handler by the appropriate method for the source language. An alternative is provided for users who do not want to modify the source program. The module LIB$ESTEMU in SYS$LIBRARY:STARLET.OLB uses LIB$INITIALIZE to enable LIB$EMULATE as a condition handler before program execution begins. To use this method, link the program with LIB$ESTEMU as follows:

```
$ LINK program, SYS$LIBRARY:STARLET/INCLUDE=LIB$ESTEMU
```

If LIB$EMULATE is established this way, the new instructions will be available to all of **program**.

Because of an addition to the VMS operating system, the functions that were performed by LIB$EMULATE are now done automatically. Therefore, although it will not cause an error to call this routine, it is never necessary to do so.

CONDITION VALUES RETURNED	SS$_RESIGNAL	Resignal condition to next handler. The exception was not one that LIB$EMULATE could handle.

LIB$ERASE_LINE Erase Line from Screen

LIB$ERASE_LINE erases all the character positions on the screen from the specified cursor position to the end of the line.

FORMAT

LIB$ERASE_LINE *[line-no ,col-no]*

RETURNS

type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

line-no

type: **word integer (signed)**
access: **read only**
mechanism: **by reference**

Number of the line at which LIB$ERASE_LINE begins erasing. The **line-no** argument is the address of a signed word integer containing this line number. If omitted, the default is the current line. This is an optional argument. However, if **line-no** is specified, **col-no** must also be specified or you will receive an invalid argument error.

col-no

type: **word integer (signed)**
access: **read only**
mechanism: **by reference**

Number of the column at which LIB$ERASE_LINE begins erasing. The **col-no** argument is the address of a signed word integer containing this column number. This is an optional argument. If omitted, the default is the current column. However, if **col-no** is specified, **line-no** must also be specified or you will receive an invalid argument error.

CONDITION VALUES RETURNED

SS$_NORMAL	Routine successfully completed.
LIB$_INVARG	Invalid argument. You must specify both aruments or no arguments.
LIB$_INVSCRPOS	Invalid screen position values. **Line-no** or **col-no** was zero.

EXAMPLE

```
ICOL = 42
ILINE = 12
ISTAT = LIB$ERASE_LINE (ILINE,ICOL)
```

This FORTRAN code fragment erases the screen from column 41 of line 12 to the end of line 12.

LIB$ERASE_PAGE Erase Page from Screen

LIB$ERASE_PAGE erases all the character positions on the screen from the specified cursor position to the end of the screen.

FORMAT

LIB$ERASE_PAGE *[line-no ,col-no]*

RETURNS

type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

line-no

type: **word integer (signed)**
access: **read only**
mechanism: **by reference**

Number of the line at which LIB$ERASE_PAGE begins erasing. The **line-no** argument is the address of a signed word integer containing this line number. This is an optional argument. If omitted, the default is the current line number. However, if **line-no** is specified, **col-no** must also be specified or you will receive an invalid argument error.

col-no

type: **word integer (signed)**
access: **read only**
mechanism: **by reference**

Number of the column at which LIB$ERASE_PAGE begins erasing. The **col-no** argument is the address of a signed word integer containing this column number. This is an optional argument. If omitted, the default is the current column number. However, if **col-no** is specified, **line-no** must also be specified or you will receive an invalid argument error.

CONDITION VALUES RETURNED

SS$_NORMAL	Routine successfully completed.
LIB$_INVARG	Invalid argument. You must specify two arguments or no arguments.
LIB$_INVSCRPOS	Invalid screen position values. **Line-no** or **col-no** was zero.

LIB$GET_SCREEN Get Text from Screen

LIB$GET_SCREEN reads an input string from SYS$INPUT into a destination text string. The destination string can be fixed-length or dynamic.

FORMAT

LIB$GET_SCREEN *input-text [,prompt-str][,out-len]*

RETURNS

type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

input-text

type: **character string**
access: **write only**
mechanism: **by descriptor**

Text copied from the screen by LIB$GET_SCREEN. The **input-text** argument is the address of a descriptor pointing to the input text.

prompt-str

type: **character string**
access: **read only**
mechanism: **by descriptor**

Prompt displayed by LIB$GET_SCREEN prior to accepting input from the user terminal. The **prompt-str** argument is the address of a descriptor pointing to the prompt string. The prompt is shown on the screen starting at the current cursor position.

out-len

type: **word (unsigned)**
access: **write only**
mechanism: **by reference**

Number of characters LIB$GET_SCREEN writes into **input-text**, not counting padding in the case of a fixed-length string. The **out-len** argument is the address of an unsigned word into which LIB$GET_SCREEN writes this number. If the input string is truncated to the size specified in the **input-text** descriptor, **out-len** is set to this size. Therefore, **out-len** can always be used by the calling program to access a valid substring of **input-text**.

CONDITION VALUES RETURNED

SS$_NORMAL	Routine successfully completed.
LIB$_INPSTRTRU	The input string is truncated to the size specified in the **input-text** descriptor.
LIB$_INVSTRDES	Invalid argment. The descriptor class field is not a recognized code.
LIB$_SCRBUFOVF	Screen buffer overflow.

Condition values returned by RMS.

LIB$PUT_BUFFER Put Current Buffer to Screen or to Previous Buffer

LIB$PUT_BUFFER terminates the current buffering mode, and reverts to the previous mode as specified by the argument **old-buffer**.

FORMAT

LIB$PUT_BUFFER *[old-buffer]*

RETURNS

type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

old-buffer

type: **longword integer (unsigned)**
access: **read only**
mechanism: **by reference**

Zero or the address of an area previously used as a screen buffer. The **old-buffer** argument is the address of an unsigned longword containing this value or address. If **old-buffer** is omitted or contains zero, the contents of the current screen buffer are displayed on the screen, the data length of the buffer is set to zero, and buffer mode is terminated. If **old-buffer** is not zero, it is assumed to be the address of an area previously used as a screen buffer. The contents of the current active buffer are copied to this area which then becomes the new active buffer.

DESCRIPTION

LIB$PUT_BUFFER terminates the current buffering mode.

If the argument is zero or omitted:

- Buffering is terminated
- The contents of the current screen buffer are displayed on the screen

If the argument is not zero:

- Buffering is terminated at the current level
- The value of the argument is taken as the address of a previous screen buffer to which the data from the current buffer is copied
- The current buffer is set to zero length
- The previous buffer becomes the active buffer

Each modular program should use LIB$SET_BUFFER and LIB$PUT_BUFFER in pairs.

LIB$PUT_BUFFER

LIB$SET_BUFFER establishes the current buffering mode and saves the address of the previous buffer (if any). LIB$PUT_BUFFER reverts from the current buffering mode to the previous mode through the use of the previous buffer address, made available by the corresponding LIB$SET_BUFFER procedure call from the current modular program.

If buffering was in effect at the time of the call to LIB$SET_BUFFER in this modular program, the contents of the current buffer are copied to the previous buffer, and the previous buffer is reestablished as the active buffer. If buffering was not in effect, buffer mode is terminated and the contents of the buffer are displayed on the terminal.

CONDITION VALUES RETURNED

SS$_NORMAL	Routine successfully completed.
LIB$_SCRBUFOVF	Screen buffer overflow.

LIB$PUT_LINE Put Text to Screen in Line Mode

LIB$PUT_LINE displays the specified text on the screen, beginning at the current cursor position followed by a specified number of line advances.

FORMAT

LIB$PUT_LINE *text [,line-adv][,flags]*

RETURNS

type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

text

type: **character string**
access: **read only**
mechanism: **by descriptor**

Text that LIB$PUT_LINE writes to the screen. The **text** argument is the address of a descriptor pointing to this text.

line-adv

type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Number of lines which LIB$PUT_LINE advances the cursor after displaying the text. The **line-adv** argument is the address of a signed longword integer containing this number. If **line-adv** is negative, the cursor moves upward after displaying the line; if it is positive, the cursor moves downward. The default value is one.

flags

type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Attributes bit vector used by LIB$PUT_LINE to determine the terminal characteristics. The **flags** argument is the address of an unsigned longword containing the attributes bit vector.

A bit must be set to one to get the desired attribute. Bits are additive. Currently, these bits are used:

Mnemomic	Bit	Description
SCR$M_BOLD	0	Bold
SCR$M_REVERSE	1	Reverse Video
SCR$M_BLINK	2	Blinking
SCR$M_UNDERLINE	3	Underscored

LIB$PUT_LINE

DESCRIPTION

LIB$PUT_LINE displays the specified text on the screen beginning at the current cursor position and followed by a specified number of line advances.

Terminal attributes, such as bold, blinking, reverse video, and underscoring, can be specified by an optional argument.

By default, the cursor moves to the beginning of the next line. The effect is the same as a carriage return and line feed.

CONDITION VALUES RETURNED

SS$_NORMAL	Routine successfully completed.
LIB$_INVARG	Invalid argument. More than three arguments were specified.

LIB$PUT_SCREEN Put Text to Screen

LIB$PUT_SCREEN displays the specified text on the screen beginning at a specified line and column. No carriage return or line feed control characters are inserted.

FORMAT

LIB$PUT_SCREEN *text [,line-no,col-no][,flags]*

RETURNS

type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

text

type: **character string**
access: **read only**
mechanism: **by descriptor**

Text that LIB$PUT_SCREEN writes to the screen. The **text** argument is the address of a descriptor pointing to this text.

line-no

type: **word integer (signed)**
access: **read only**
mechanism: **by reference**

Number of the line at which LIB$PUT_SCREEN begins writing the text. The **line-no** argument is the address of a signed word integer containing this line number. This is an optional argument. If omitted, the default is the current line.

col-no

type: **word integer (signed)**
access: **read only**
mechanism: **by reference**

Number of the column at which LIB$PUT_SCREEN begins writing the text. The **col-no** argument is the address of a signed word integer containing this column number. This is an optional argument. If omitted, the default is the current column.

flags

type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Attributes bit vector which LIB$PUT_SCREEN uses to determine the terminal characteristics. The **flags** argument is the address of an unsigned longword containing these flags.

A bit must be set to one to get the desired attribute. Bits are additive. Currently, these bits are used:

Mnemomic	Bit	Description
SCR$M_BOLD	0	Bold
SCR$M_REVERSE	1	Reverse Video
SCR$M_BLINK	2	Blinking
SCR$M_UNDERLINE	3	Underscored

DESCRIPTION

LIB$PUT_SCREEN displays the specified text on the screen beginning at a specified line and column. No carriage return or line feed control characters are inserted.

Terminal attributes, such as bold, blinking, reverse video, and underscoring, can be specified by an optional argument, **flags**. See Table 3–3 for the possible values for the **flags** argument.

CONDITION VALUES RETURNED

SS$_NORMAL	Routine successfully completed.
LIB$_INVARG	Invalid argument. The call to LIB$PUT_SCREEN specified more than four arguments.
LIB$_INVSCRPOS	Invalid screen position values. **Line-no** or **col-no** was zero.

LIB$SCREEN_INFO Retrieve Screen Information

LIB$SCREEN_INFO returns a terminal's characteristics, including flags, device type, screen width, number of lines per screen, and special characteristics, to your program.

FORMAT

LIB$SCREEN_INFO *flags [,dev-type] [,line-width] [,lines-per-page]*

RETURNS

type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

flags

type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

Bit map representing special terminal characteristics. The **flags** argument is the address of an unsigned longword containing these flags.

Currently, these bits are used:

Bit	Description
0	The terminal is a DIGITAL video terminal.
1	The terminal is a VT100-compatible terminal.
2	The terminal understands REGIS graphic commands.
3	The terminal is a block mode terminal.
4	The terminal has the standard VT100 AVO.

dev-type

type: **byte integer (signed)**
access: **write only**
mechanism: **by reference**

Terminal type. The **dev-type** argument is the address of a signed byte integer containing the device type. The terminal types are defined in the VAX System Services $DCDEF macro.

Some of the terminal types are:

Type	Equivalent DCL Command
Unknown type	SET TERMINAL/DEVICE_TYPE=UNKNOWN
VT52	SET TERMINAL/DEVICE_TYPE=VT52
VT100	SET TERMINAL/DEVICE_TYPE=VT100

line-width

type: **word integer (signed)**
access: **write only**
mechanism: **by reference**

Width (in columns) for which the terminal is configured. The **line-width** argument is the address of a signed word integer containing the line width. This corresponds to the value supplied by the DCL command SET TERMINAL/WIDTH=n.

lines-per-page

type: **word integer (signed)**
access: **write only**
mechanism: **by reference**

Lines (per screen) for which the terminal is configured. The **lines-per-page** argument is the address of a signed word integer containing the number of lines per screen. This corresponds to the value supplied by the DCL command SET TERMINAL/PAGE=n.

CONDITION VALUES RETURNED

None.

LIB$SET_BUFFER Set or Clear Screen Buffer Mode

LIB$SET_BUFFER establishes or ends screen buffering.

FORMAT

LIB$SET_BUFFER *buffer [,old-buffer]*

RETURNS

type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

buffer

type: **character sring**
access: **modify**
mechanism: **by descriptor, fixed-length**

Modifiable fixed-length string used as the buffer for storing characters that the other screen output procedures would normally send to the terminal without buffering until the next LIB$SET_BUFFER or LIB$PUT_BUFFER procedure call occurs. The **buffer** argument is the address of a descriptor pointing to this buffer. The buffer must be at least 12 bytes long.

If **buffer** is zero, buffering mode is terminated.

old-buffer

type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

Previous buffer (if any). The **old-buffer** argument is the address of a longword integer containing the previous buffer. The **old-buffer** is most useful for subsequent use as an input argument to LIB$PUT_BUFFER.

DESCRIPTION

LIB$SET_BUFFER is called by your program to set up a buffer and initiate the buffering mode. While in buffer mode, the other screen procedures do not alter the appearance of the screen. Instead, the output of these procedures is stored in the buffer set up by LIB$SET_BUFFER.

To display the results of all procedures called since the buffering mode was entered, your program must call LIB$PUT_BUFFER.

It is sometimes useful, when writing a program with subprogams, to have more than one buffer initialized at the same time. If you call LIB$SET_BUFFER a second time, without having called LIB$PUT_BUFFER, a second buffer will be set up. When using LIB$SET_BUFFER to set up a second buffer, you MUST specify the **old-buffer** argument. LIB$SET_BUFFER will write the address of the old buffer into this argument. Because the second buffer is generally set up for use within a subprogram, it is good practice to append the contents of this subprogram buffer to the end of the mainprogram buffer at the end of the subprogram call. This will ensure that when the

contents of the buffer are written to the screen, the displays will appear in their original order. This is also the reason that LIB$SET_BUFFER writes the address of that mainprogram buffer into the **old-buffer** argument. To append the contents of your subprogram buffer to the end of the mainprogram buffer, your program must call LIB$PUT_BUFFER and pass the routine to the **old-buffer** argument.

When setting up the buffer, it is important to make sure that the buffer will be large enough to hold the header information, and the terminal commands. You must allow 12 bytes within the buffer for header information. If your buffer size is less than 12 bytes, LIB$SET_BUFFER will return immediately with the error LIB$_SCRBUFOVF, screen buffer overflow.

If your buffer size is greater than 12 bytes, you may still get the LIB$_SCRBUFOVF error if your program attempts to write text to the buffer which is larger than the size of the buffer.

The MAXBUF system argument will determine how large a buffer you may declare. Although MAXBUF varies slightly from system to system, a typical size is 1500 bytes. Your buffer may not exceed the size of MAXBUF.

CONDITION VALUES RETURNED

SS$_NORMAL	Routine successfully completed.
LIB$SCRBUFOVF	Screen buffer overflow. The buffer is less than 12 bytes long.
LIB$_INVARG	Invalid argument. Zero or more than two arguments were specified.

LIB$SET_CURSOR Set Cursor to Character Position on Screen

LIB$SET_CURSOR positions the cursor to the specified line and column on the screen.

FORMAT

LIB$SET_CURSOR *line-no ,col-no*

RETURNS

type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

ARGUMENTS

line-no

type: **word integer (signed)**
access: **read only**
mechanism: **by reference**

Number of the line to which LIB$SET_CURSOR moves the cursor. The **line-no** argument is the address of a signed word integer containing this line number.

col-no

type: **word integer (signed)**
access: **read only**
mechanism: **by reference**

Number of the column to which LIB$SET_CURSOR moves the cursor. The **col-no** argument is the address of a signed word integer containing this column number.

DESCRIPTION

LIB$SET_CURSOR lets your program control the cursor's position on the screen without knowing what type of DIGITAL terminal will be used. If the terminal is unable to change the cursor's position, it ignores the procedure call and continues without issuing an error message.

LIB$SET_CURSOR considers the screen's top line is line number one and that its leftmost column is column number one. If you give a value of zero for either of these arguments, you will receive the error LIB$_INVARG.

Since the terminal-independent screen procedures are designed to display the information you have passed to them as accurately as possible, they ignore formatting errors. Thus, LIB$SET_CURSOR does not check for cursor position specifications that exceed the maximum number of lines or columns for the terminal.

CONDITION VALUES RETURNED

SS$_NORMAL	Routine successfully completed.
LIB$_INVARG	Invalid argument. LIB$SET_CURSOR requires exactly two arguments.
LIB$_INVSCRPOS	Invalid screen position values. **Line-no** or **col-no** was zero.

EXAMPLE

```
ISTAT = LIB$SET_CURSOR (5, 7)
```

This FORTAN code fragment moves the cursor to column seven of line five.

LIB$SET_OUTPUT Set a Terminal or a Screen Buffer for Output

LIB$SET_OUTPUT establishes a terminal or buffer to receive output.

FORMAT

LIB$SET_OUTPUT *[stream] [,file-spec] [,user-routine] [,user-arg] [,old-stream]*

RETURNS

type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

stream

type: **word integer (signed)**
access: **read only**
mechanism: **by reference**

Stream number, assigned by the caller. The **stream** argument is the address of a signed word integer pointing to the stream number. This is an optional argument. If omitted, the default is 0, SYS$OUTPUT.

file-spec

type: **character string**
access: **read only**
mechanism: **by descriptor**

File specification of the file to which you want LIB$SET_OUTPUT to send the output. The **file-spec** argument is the address of a descriptor pointing to the file specification. This is an optional argument.

user-routine

type: **procedure entry mask**
access: **read only**
mechanism: **by reference, procedure reference**

User-supplied routine. The **user-routine** argument is the address of the entry mask to the user routine.

By default, LIB$SET_OUTPUT uses RMS to open the output file. You can open the file yourself using a user routine. The user routine is also called whenever LIB$SET_OUTPUT needs to send output to the device or file. Normally, LIB$SET_OUTPUT does the $QIOs to terminals and the $PUTs to files. The user routine allows you to do these yourself. For more information on the user routine, see "Call Format for a User Routine" in the Description Section.

user-arg

type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

32-bit value passed to the user routine without interpretation. The **user-arg** argument is the address of a signed longword integer containing the user argument.

old-stream

type: **longword integer (signed)**
access: **write only**
mechanism: **by reference**

Previous stream number. The **old-stream** argument is the address of a signed longword integer containing the previous stream number.

DESCRIPTION

This Description section has been divided into two parts:

- Guidelines for Using LIB$SET_OUTPUT
- Call Format for a User Routine

Guidelines for Using LIB$SET_OUTPUT

LIB$SET_OUTPUT allows you to direct output to a terminal or file other than the default output device, SYS$OUTPUT. If you are sending output to only one device, SYS$OUTPUT, the call to LIB$SET_OUTPUT is optional. However, by using LIB$SET_OUTPUT a single process can display information on several terminals at the same time. To do this, your program calls LIB$SET_OUTPUT once for each terminal, before it sends any output to the terminals.

You can also use LIB$SET_OUTPUT to direct output to a file. By default, LIB$SET_OUTPUT uses RMS to open the file, but you can include a user routine to open the file yourself and output to the file. You also can perform your own $QIOs to a terminal via this user routine.

The first time you call LIB$SET_OUTPUT, you must use all the arguments. This sets up the channel to the device or file. On this initial call, LIB$SET_OUTPUT obtains the device characteristics. If the device type is unknown, the channel is deassigned. After this, you can use just the file specification or device name along with the user routine and user argument for each call.

If you call LIB$SET_OUTPUT with no arguments, the output goes to SYS$OUTPUT, the default process output stream.

LIB$STOP_OUTPUT can be used to close the output stream established by LIB$SET_OUTPUT. If you do not call LIB$STOP_OUTPUT, the channel will be freed automatically when the image exits. However, if you do call LIB$STOP_OUTPUT, you MUST call LIB$SET_OUTPUT afterward to resume output even if you are using the default, SYS$OUTPUT. If you do not follow a call to LIB$STOP_OUTPUT with a call to LIB$SET_OUTPUT, you will get a LIB$INVCHA error.

You should use LIB$STOP_OUTPUT if you wish to make multiple versions of a single file. Otherwise, if you call LIB$SET_OUTPUT several times to send output to a file, the original file will be overwritten and all the output except the last version will be lost. If you call LIB$STOP_OUTPUT after each call to LIB$SET_OUTPUT, however, LIB$SET_OUTPUT will open a new version of the file each time, and all of the output will be preserved.

Call Format for a User Routine

LIB$SET_OUTPUT calls the user routine using the format:

```
user-routine  [,user-arg] [,chan] [,output-string] [,stream]
```

Arguments

user-arg

type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

32-bit value that LIB$SET_OUTPUT passes to your user routine without interpretation. The **user-arg** argument is the address of a signed longword integer containing the user argument.

chan

type: **word integer (signed)**
access: **read only**
mechanism: **by reference**

Channel or stream number. The **chan** argument is the address of a signed word integer pointing to the channel.

output-string

type: **character string**
access: **read only**
mechanism: **by descriptor**

Output string passed to your user routine by LIB$SET_OUTPUT. The **output-string** argument is the address of a descriptor pointing to the output string. This is an optional argument.

stream

type: **longword integer (signed)**
access: **write only**
mechanism: **by reference**

Previous stream number. The **stream** argument is the address of a signed longword integer containing the previous stream number.

CONDITION VALUES RETURNED

SS$_NORMAL	Routine successfully completed.

Condition values returned by your user routine.

LIB$SET_SCROLL Set Scrolling Region

LIB$SET_SCROLL establishes a scrolling region by setting the internal scrolling region arguments. The cursor position is unchanged.

FORMAT

LIB$SET_SCROLL *start-line ,end-line*

RETURNS

type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

start-line

type: **word integer (signed)**
access: **read only**
mechanism: **by reference**

Starting line of the scrolling region that LIB$SET_SCROLL establishes. The **start-line** argument is the address of a signed word integer containing the starting line.

end-line

type: **word integer (signed)**
access: **read only**
mechanism: **by reference**

Ending line of the scrolling region LIB$SET_SCROLL establishes. The **end-line** argument is the address of a signed word integer containing the ending line.

CONDITION VALUES RETURNED

SS$_NORMAL	Routine successfully completed.
LIB$_INVARG	Invalid arguments. You must specify exactly two arguments to LIB$SET_SCROLL.

LIB$STOP_OUTPUT Stop Output to a Terminal or Screen Buffer

LIB$STOP_OUTPUT deassigns a terminal or buffer stream established for output.

FORMAT

LIB$STOP_OUTPUT *[chan]*

RETURNS

type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

chan

type: **word integer (signed)**
access: **read only**
mechanism: **by reference**

Channel number that LIB$STOP_OUTPUT deassigns. The **chan** argument is the address of a signed word integer containing this number. This argument is currently not used.

DESCRIPTION

LIB$STOP_OUTPUT can be used to close an output stream established by LIB$SET_OUTPUT. If you do not call LIB$STOP_OUTPUT, the channel will be freed automatically when the image exits.

LIB$STOP_OUTPUT is useful, for example, if you wish to make multiple versions of a single file. If you call LIB$SET_OUTPUT several times to send output to a file, and do not call LIB$STOP_OUTPUT, the original file will be overwritten. All of the output execept the last version will be lost. If you call LIB$STOP_OUTPUT after each call to LIB$SET_OUTPUT, however, LIB$SET_OUTPUT will open a new version of the file each time, and all of the output will be preserved.

If the device type is unknown, the channel is deassigned. If a file specification is given but no user output routine is declared, then the file is opened using RMS.

CONDITION VALUES RETURNED

SS$_NORMAL	Routine completed successfully.

LIB$UP_SCROLL Up Scroll, Move Cursor Down One Line

LIB$UP_SCROLL moves the cursor down one line on the screen.

FORMAT

LIB$UP_SCROLL

RETURNS

type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

None.

DESCRIPTION

LIB$UP_SCROLL moves the cursor down one line on the screen. If the cursor was already at the bottom line of the screen, all lines are moved up one line. The information that was on the top line is lost and a blank line appears at the bottom.

If a scrolling region is active, then the above logic applies to the top and bottom lines of the scrolling region.

CONDITION VALUES RETURNED

SS$_NORMAL	Routine completed successfully.

SCR$DOWN_SCROLL Down Scroll, Move Cursor Up One Line

SCR$DOWN_SCROLL moves the cursor up one line on the screen.

FORMAT

SCR$DOWN_SCROLL

RETURNS

type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

None.

DESCRIPTION

If the cursor is already at the top line on the screen, SCR$DOWN_SCROLL moves all lines down one line, and replaces the top line with a blank line. Any data that was on the bottom line is lost.

If a scrolling region is active, then the above logic applies to the top and bottom lines of the scrolling region.

CONDITION VALUES RETURNED

SS$_NORMAL	Routine successfully completed.

SCR$ERASE_LINE Erase Line from Screen

SCR$ERASE_LINE erases all of the character positions on the screen from the specified cursor position to the end of the line.

FORMAT

SCR$ERASE_LINE *[line-no ,col-no]*

RETURNS

type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

line-no

type: **word integer (signed)**
access: **read only**
mechanism: **by value**

Number of the line at which SCR$ERASE_LINE begins erasing. The **line-no** argument is a signed word integer containing the line number. This is an optional argument; however, if it is specified, the **col-no** argument must also be specified. If **line-no** is omitted or if **line-no** is specified but **col-no** is omitted, SCR$ERASE_LINE uses the current line number.

col-no

type: **word integer (signed)**
access: **read only**
mechanism: **by value**

Number of the column at which SCR$ERASE_LINE begins erasing. The **col-no** argument is a signed word integer containing the column number. This is an optional argument; however, if it is specified, the **line-no** argument must also be specified. If **col-no** is omitted or if **col-no** is specified but **line-no** is omitted, SCR$ERASE_LINE uses the current column number.

CONDITION VALUES RETURNED

SS$_NORMAL	Routine successfully completed.

SCR$ERASE_PAGE Erase Page from Screen

SCR$ERASE_PAGE erases all of the character positions on the screen from the specified cursor position to the end of the screen.

FORMAT

SCR$ERASE_PAGE *[line-no ,col-no]*

RETURNS

type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

line-no

type: **word integer (signed)**
access: **read only**
mechanism: **by value**

Number of the line at which SCR$ERASE_PAGE begins erasing. The **line-no** argument is a signed word integer containing the line number. This is an optional argument; however, if it is specified, the **col-no** argument must also be specified. If **line-no** is omitted or if **line-no** is specified but **col-no** is omitted, SCR$ERASE_PAGE uses the current line number.

col-no

type: **word integer (signed)**
access: **read only**
mechanism: **by value**

Number of the column at which SCR$ERASE_PAGE begins erasing. The **col-no** argument is a signed word integer containing the column number. This is an optional argument; however, if it is specified, the **line-no** argument must also be specified. If **col-no** is omitted or if **col-no** is specified but **line-no** is omitted, SCR$ERASE_PAGE uses the current column number.

CONDITION VALUES RETURNED

SS$_NORMAL	Routine successfully completed.

SCR$GET_SCREEN Get Text from Screen

SCR$GET_SCREEN reads an input string from SYS$INPUT into a destination text string. The destination string can be fixed-length or dynamic.

FORMAT

SCR$GET_SCREEN *input-text [,prompt-str] [,out-len]*

RETURNS

type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

input-text

type: **character string**
access: **write only**
mechanism: **by descriptor**

Input text string into which SCR$GET_SCREEN writes the text copied from the screen. The **input-text** argument is the address of a descriptor pointing to the input text string.

prompt-str

type: **charcter string**
access: **read only**
mechanism: **by descriptor**

String that SCR$GET_SCREEN displays prior to accepting input from the user terminal. The **prompt-str** argument is the address of a descriptor pointing to the prompt string. The string is shown on the screen starting at the current cursor position.

out-len

type: **word (unsigned)**
access: **write only**
mechanism: **by reference**

Number of characters written into **input-text**, not counting padding in the case of a fixed-length string. The **out-len** argument is the address of the number of characters. If the input string is truncated to the size specified in the **input-text** descriptor, **out-len** is set to this size. Therefore, **out-len** can always be used by the calling program to access a valid substring of **input-text**.

CONDITION VALUES RETURNED

SS$_NORMAL	Routine completed successfully.
LIB$_INPSTRTRU	The input string is truncated to the size specified in the **input-text** descriptor.
LIB$INSVIRMEM	Insufficent virtual memory.
LIB$_INVARG	Invalid argument. Descriptor class field is not a recognized code or it is zero.
RMS$_xyz	Condition values returned by RMS.

SCR$SCREEN_INFO Screen Information Retrieval

SCR$SCREEN_INFO moves terminal specifications to the area or areas you specify.

FORMAT

SCR$SCREEN_INFO *control-block*

RETURNS

type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

control-block
type: **longword integer (signed)**
access: **write only**
mechanism: **by reference**

Nine bytes which correspond, in order, to the **flags, line-width, lines-per-page**, and **dev-type** arguments specified for LIB$SCREEN_INFO. The **control-block** argument is the address of this information.

CONDITION VALUES RETURNED

SS$_NORMAL	Procedure successfully completed.
LIB$_NO_STRACT	No active stream on which to return data.

SCR$PUT_BUFFER Put Current Buffer to Screen or to Previous Buffer

SCR$PUT_BUFFER terminates the current buffering mode.

FORMAT

SCR$PUT_BUFFER *[old-buffer]*

RETURNS

type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

old-buffer

type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

Address of the area previously used as a screen buffer. The **old-buffer** argument is the address of that address. This is an optional argument. If **old-buffer** is omitted or contains zero, the contents of the current screen buffer are displayed on the screen, the data length of the buffer is set to zero, and the buffer mode is terminated. If **old-buffer** is not zero, it is assumed to be the address of an area previously used as a screen buffer. The contents of the current active buffer are copied to this area which then becomes the new active buffer.

DESCRIPTION

If the address of the previous screen buffer is given, SCR$PUT_BUFFER copies the contents of the current buffer to that address before it terminates the current buffer. If the **old-buffer** argument was not specified, SCR$PUT_BUFFER empties the buffer to the terminal screen.

CONDITION VALUES RETURNED

SS$_NORMAL	Routine successfully completed.
LIB$_SCRBUFOVF	Screen buffer overflow.

SCR$PUT_LINE Put Text to Screen in Line Mode

SCR$PUT_LINE displays the specified text on the screen, beginning at the current cursor position followed by a specified number of line advances.

Terminal attributes, such as bold, blinking, reverse video, and underscoring, can be specified by an optional argument.

FORMAT

SCR$PUT_LINE *text [,line-adv][,flags]*

RETURNS

type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

text

type: **character string**
access: **read only**
mechanism: **by descriptor**

Character string that SCR$PUT_LINE writes to the screen. The **text** argument is the address of a descriptor pointing to this character string.

line-adv

type: **longword integer (signed)**
access: **read only**
mechanism: **by value**

Number of lines that SCR$PUT_LINE advances after displaying the text. The **line-adv** argument is a signed word integer containing the number of lines to be advanced. This is an optional argument. If omitted, the default is zero.

flags

type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

Attributes bit vector. The **flags** argument an unsigned longword containing the flags.

A bit must be set to one to get the desired attribute. Bits are additive. Currently, these bits are used:

Mnemomic	Bit	Description
SCR$M_BOLD	0	Bold
SCR$M_REVERSE	1	Reverse Video
SCR$M_BLINK	2	Blinking
SCR$M_UNDERLINE	3	Underscored

This is an optional argument. If omitted, the default is SCR$M_NORMAL, and all bits are clear.

CONDITION VALUES RETURNED

SS$_NORMAL	Routine successfully completed.
LIB$_WRONUMARG	Wrong number of arguments.

SCR$PUT_SCREEN Put Text to Screen

SCR$PUT_SCREEN displays the specified text on the screen beginning at a specified line and column.

FORMAT

SCR$PUT_SCREEN *text [,line-no ,col-no] [,flags]*

RETURNS

type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

text

type: **character string**
access: **read only**
mechanism: **by descriptor**

Character string that SCR$PUT_SCREEN writes to the screen. The **text** argument is the address of a descriptor pointing to the character string.

line-no

type: **longword integer (signed)**
access: **read only**
mechanism: **by value**

Number of the line at which SCR$PUT_SCREEN begins writing the text. The **line-no** argument is a signed longword integer containing the line number. This is an optional argument; however, if it is specified, the **col-no** argument must also be specified. If **line-no** is omitted, or if **line-no** is specified but **col-no** is omitted, SCR$PUT_SCREEN uses the current line number.

col-no

type: **longword integer (signed)**
access: **read only**
mechanism: **by value**

Number of the column at which SCR$PUT_SCREEN begins writing the text. The **col-no** argument is a signed longword integer containing the column number. This is an optional argument; however, if it is specified, the **line-no** argument must also be specified. If **col-no** is omitted or if **col-no** is specified but **line-no** is omitted, SCR$PUT_SCREEN uses the current column number.

flags

type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

Attributes bit vector. The **flags** argument an unsigned longword containing the flags.

A bit must be set to one to get the desired attribute. Bits are additive. Currently, these bits are used:

Mnemomic	Bit	Description
SCR$M_BOLD	0	Bold
SCR$M_REVERSE	1	Reverse Video
SCR$M_BLINK	2	Blinking
SCR$M_UNDERLINE	3	Underscored

This is an optional argument. If omitted, the default is SCR$M_NORMAL, and all bits are clear.

DESCRIPTION

SCR$PUT_SCREEN allows terminal attributes, such as bold, blinking, reverse video, and underscoring, can be specified by an optional argument. SCR$PUT_SCREEN does not insert carriage return or line feed control characters. In a non-buffer mode, the total number of characters output is 512, including text, cursor positioning, and video attribute terminal commands.

CONDITION VALUES RETURNED

SS$_NORMAL	Routine successfully completed.
LIB$_INVARG	Invalid argument. Undefined bits set in **flags**.
LIB$_FATERRLIB	Fatal internal error in library.
LIB$_SCRBUFOVF	Screen buffer overflow. The length of the text exceeds the size of the buffer. Applicable in buffering mode only.

SCR$SET_BUFFER Set or Clear Screen Buffer Mode

SCR$SET_BUFFER sets or clears buffer mode for the other terminal independent screen procedures.

FORMAT

SCR$SET_BUFFER *buffer [,old-buffer]*

RETURNS

type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

buffer

type: **character string**
access: **modify**
mechanism: **by descriptor, fixed-length**

Buffer which SCR$SET_BUFFER sets up. The **buffer** argument is the address of a descriptor pointing to this buffer. If **buffer** is zero, SCR$SET_BUFFER terminates the buffering mode and clears the buffer.

old-buffer

type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

Address of the previous buffer (if any), supplied by SCR$SET_BUFFER. The **old-buffer** argument is the address of the address of this previous buffer. The **old-buffer** argument is most useful for subsequent use as an input argument to LIB$PUT_BUFFER.

DESCRIPTION

SCR$SET_BUFFER is called by your program to set up a buffer and initiate buffer mode. While in buffer mode, the other screen procedures do not alter the apperarance of the screen. Instead, the output of these procedures is stored in the buffer set up by SCR$SET_BUFFER.

To display the results of all procedures called since the buffering mode was entered, your program must call SCR$PUT_BUFFER.

It is sometimes useful, when writing a program with subprograms, to have more than one buffer initialized at the same time. If you call SCR$SET_BUFFER a second time without having called SCR$PUT_BUFFER, a second buffer will be set up. When using SCR$SET_BUFFER to set up a second buffer, you MUST specify the **old-buffer** argument. SCR$SET_BUFFER will write the address of the old buffer into this argument. Because the second buffer is generally set up for use within a subprogram, it is good practice to append the contents of this subprogram buffer to the end of the mainprogram buffer at the end of the subprogram call. This will ensure that when the contents of the buffer are written to the screen, the displays will appear in

their original order. This is also the reason that SCR$SET_BUFFER writes the address of that mainprogram buffer into the **old-buffer** argument. To append the contents of your subprogram buffer to the end of the mainprogram buffer, your program must call SCR$PUT_BUFFER and pass the routine the **old-buffer** argument.

When setting up the buffer, it is important to make sure that the buffer will be large enough to hold the header information and the terminal commands. You must allow 12 bytes within the buffer for header information. If your buffer size is less than 12 bytes, SCR$SET_BUFFER will return immediately with the error LIB$_SCRBUFOVF, screen buffer overflow.

If your buffer size is greater than 12 bytes, you may still get the LIB$_SCRBUFOVF error if your program attempts to write text to the buffer which is larger than the size of the buffer.

The MAXBUF system argument will determine how large a buffer you may declare. Although MAXBUF varies slightly from system to system, a typical size is 1500 bytes. Your buffer may not exceed the size of MAXBUF.

CONDITION VALUES RETURNED

SS$_NORMAL	Routine successfully completed.
LIB$_SCRBUFOVF	Screen buffer overflow. The buffer is less than 12 bytes in length.
LIB$_INVARG	Invalid argument. Zero or more than two arguments were specified.

SCR$SET_CURSOR Set Cursor to Character Position on Screen

SCR$SET_CURSOR positions the cursor to the specified line and column on the screen.

FORMAT

SCR$SET_CURSOR *line-no ,col-no*

RETURNS

type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

line-no

type: **word integer (signed)**
access: **read only**
mechanism: **by value**

Number of the line to which SCR$SET_CURSOR moves the cursor. The **line-no** argument is a signed word integer containing the line number.

col-no

type: **word integer (signed)**
access: **read only**
mechanism: **by value**

Number of the column to which SCR$SET_CURSOR moves the cursor. The **col-no** argument is a signed word integer containing the column number.

CONDITION VALUES RETURNED

SS$_NORMAL	Routine successfully completed.
LIB$_INVARG	Invalid argument. The number of arguments specified must be two.

SCR$SET_OUTPUT Set a Terminal or Screen Buffer for Output

SCR$SET_OUTPUT establishes a terminal or buffer to receive output.

FORMAT

SCR$SET_OUTPUT *[stream] [,file-spec] [,user-routine] [,user-arg] [,old-stream]*

RETURNS

type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

stream

type: **word integer (signed)**
access: **read only**
mechanism: **by value**

Stream number assigned by the caller to identify the output stream. The **stream** argument is a word integer containing the stream number value.

file-spec

type: **character string**
access: **read only**
mechanism: **by descriptor**

File specification of the file into which SCR$SET_OUTPUT directs the output. The **file-spec** argument is the address of a descriptor pointing to the file specification. This is an optional argument. If omitted, the SCR$SET_OUTPUT uses the default SYS$OUTPUT.

user-routine

type: **procedure entry mask**
access: **function call (before return)**
mechanism: **by reference, procedure reference**

User-supplied routine. The **user-routine** argument is the address of the entry mask to this routine. By default, SCR$SET_OUTPUT uses RMS to open the output file. You can open the file yourself using the user routine. The user routine is also called whenever the screen routine needs to send output to the device or file. Normally, the screen routine does the $QIOs to terminals and $PUTs to files. The user routine allows you to do these yourself. For more information on the user routine, see "Call Format for a User Routine" in the Description Section.

user-arg

type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Optional 32-bit value that SCR$SET_OUTPUT stores for the user routine. The **user-arg** argument is the address of the user argument.

stream

type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Previous stream number. The **stream** argument is the address of the previous stream number. This is an optional argument.

DESCRIPTION

This Description Section has been divided into two parts:

1 Guidelines for Using SCR$SET_OUTPUT

2 Call Format for a User Routine

Guidelines for Using SCR$SET_OUTPUT

SCR$SET_OUTPUT allows you to direct output to a terminal or file other than the default output device, SYS$OUTPUT. If you are sending output to only one device, SYS$OUTPUT, the call to SCR$SET_OUTPUT is optional. However, by using SCR$SET_OUTPUT a single process can display information on several terminals at the same time. To do this, your program calls SCR$SET_OUTPUT once for each terminal, before it sends any output to the terminals.

You can also use SCR$SET_OUTPUT to direct output to a file. By default, SCR$SET_OUTPUT uses RMS to open the file, but you can include a user routine to open the file yourself and output to the file. You also can perform your own $QIOs to a terminal via this user routine.

The first time you call SCR$SET_OUTPUT, you must use all the arguments. This sets up the channel to the device or file. On this initial call, SCR$SET_OUTPUT obtains the device characteristics. If the device type is unknown, the channel is deassigned. After this, you can use just the file specification or device name along with the user routine and user argument for each call.

If you call SCR$SET_OUTPUT with no arguments, the output goes to SYS$OUTPUT, the default process output stream.

Call Format for a User Routine

At output time, the user-supplied routine, if present, is called by the main program, using the address left to it by SCR$SET_OUTPUT. The user routine is called using the format:

```
user-routine   user-arg ,chan ,output-string ,stream
```

Arguments

user-arg

type: **unspecified**
access: **read only**
mechanism: **by reference**

32-bit value that was stored by SCR$SET_OUTPUT, or zero if none was supplied to SCR$SET_OUTPUT. The **user-arg** argument is the address of the user argument.

chan

type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Channel number assigned internally in order to perform QIOs, or zero if there are no QIOs to be performed. The **chan** argument is the address of an unsigned longword integer pointing to the channel number.

output-string

type: **character string**
access: **read only**
mechanism: **by descriptor**

String to be output. The **output-string** argument is the address of a descriptor pointing to the output string.

stream

type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Stream number of current output stream. The **stream** argument is the address of a signed longword integer containing the stream number.

CONDITION VALUES RETURNED

SS$_NORMAL	Routine successfully completed.

Condition values returned by your user routine.

SCR$SET_SCROLL Set Scrolling Region

SCR$SET_SCROLL establishes a scrolling region by setting the internal scrolling region arguments. The cursor position is unchanged.

FORMAT

SCR$SET_SCROLL *start-line ,end-line*

RETURNS

type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

start-line

type: **word (unsigned)**
access: **read only**
mechanism: **by value**

First line of the scrolling region. The **start-line** argument is an unsigned word containing the number of the first line.

end-line

type: **word (unsigned)**
access: **read only**
mechanism: **by value**

Last line of the scrolling region. The **end-line** argument is an unsigned word containing the number of the last line.

CONDITION VALUES RETURNED

SS$_NORMAL	Routine successfully completed.

SCR$STOP_OUTPUT Stop Output to a Terminal or Screen Buffer

SCR$STOP_OUTPUT deassigns a terminal or buffer stream established for output.

FORMAT

SCR$STOP_OUTPUT *chan*

RETURNS

type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

chan

type: **word integer (signed)**
access: **read only**
mechanism: **by value**

Channel number to be deassigned. The **chan** argument is a signed word integer containing the channel number. This argument is not currently used and must contain zero.

CONDITION VALUES RETURNED

SS$_NORMAL	The routine completed successfully.

SCR$UP_SCROLL Up Scroll, Move Cursor Down One Line

SCR$UP_SCROLL moves the cursor down one line on the screen.

FORMAT

SCR$UP_SCROLL

RETURNS

type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

None.

DESCRIPTION

SCR$UP_SCROLL moves the cursor down one line on the terminal screen. If the cursor is already at the bottom line of the screen, SCR$UP_SCROLL moves all lines up one line. The information that was on the top line is lost and a blank line appears at the bottom.

If a scrolling region is active, then the above logic applies to the top and bottom lines of the scrolling region.

CONDITION VALUES RETURNED

SS$_NORMAL	The routine completed successfully.

4 Obsolete Utilities and Utility Components

This chapter describes obsolete utilities. The first part of the chapter describes the Disk Quota Utility, which has been superseded by the System Management (SYSMAN) Utility. The second part of the chapter contains tables and descriptions of obsolete components in currently supported VMS utilities.

An *obsolete utility* is a VMS utility that has been superseded by one that is more flexible or more efficient. Obsolete utilities are no longer updated. An "obsolete utility component" is a component that has been replaced in a currently supported VMS utility.

4.1 Obsolete Utilities

This section contains a description of the Disk Quota Utility which has been replaced by the System Management (SYSMAN) Utility.

DISK QUOTA

The Disk Quota Utility (DISKQUOTA) allows you to allocate disk space to users and maintain an accurate record of disk use for ODS Level 2 disks.

4.1.1 Establishing Disk Quotas

Using the Disk Quota Utility, you create a quota file for each volume or each volume set. The quota file records all users allowed to use the disk and shows their current disk usage as well as their maximum disk allocation. A quota file has the following format:

UIC	Usage	Permanent Quota	Overdraft Limit
[0,0]	0	333333	3333
[TTD,DAVIS]	15590	333333	3333
[TTD,MORGAN]	1929	333333	3333
[MKT,MORSE]	7650	333333	3333

- The User Identification Code (UIC) of each user entitled to maintain files on the volume appears in the UIC column.
- The number of disk blocks currently dedicated to a user's files appears in the Usage column.
- The maximum number of blocks on the volume that a user's files can consume appears in the Permanent Quota column. Once exceeded, the system issues an error message.
- The number of blocks over the quota that a user's files can consume appears in the Overdraft Limit column.

The absolute maximum number of blocks permitted a user on a volume is the sum of the quota and the overdraft. Only users with the EXQUOTA privilege can bypass the quota file, but even then, their names are listed in the file without a maximum allocation.

The quota file, called QUOTA.SYS, is stored in directory [000000] with other system files and requires one block of disk storage for each 16 entries.

DIGITAL recommends that you do not enable disk quotas on the system volume.

4.1.2 Creating a Quota File

Whether building a quota file for a newly-formatted disk or a disk with existing files, first invoke the utility and, if necessary, designate the disk volume for which you are building a quota file. The Disk Quota Utility uses your current default disk, SYS$DISK, unless you specify another volume with the USE command.

To establish the quota file, enter the CREATE command. The utility sets up the file QUOTA.SYS in the directory [000000], and inserts an entry called UIC [0,0]. You then use UIC [0,0] to initialize default values for permanent quotas and overdraft limits, with the command MODIFY /PERMQUOTA=n /OVERDRAFT=n [0,0].

Finally, for disks with existing files, enter the REBUILD command so the utility reads the disk and enters all UICs and their current disk usage into the quota file. For a newly created disk, use the ADD command to add individual UICs to the quota file.

4.1.3 Maintaining a Quota File

During normal use of a volume with a quota file, the system automatically updates the usage counts as users create, delete, extend, and truncate files.

When a user creates new files, usage counts must be below the permanent quota. If an operation to add new files or expand a current file exceeds the usage count quota, the system prohibits the operation and issues an error message. A user can successfully perform a write operation, even if over quota, by trying it a second time. Operations to extend the file succeed until the usage exceeds the sum of the quota and the overdraft values.

4.1.4 Disabling a Quota File

The DISABLE command suspends quota operations on a volume. The ENABLE command lifts the suspension, at which point it is necessary to rebuild the quota file to update UICs and usage counts.

4.1.5 Listing of Commands

Table 4–1 summarizes the DISKQUOTA commands by format and function. See the Command Section for a complete description of commands.

Table 4–1 DISKQUOTA Command Summary

Command	Function
ADD	Adds an entry to the quota file
CREATE	Creates a quota file for a volume that does not currently contain one
DISABLE	Suspends quota operations on a volume
ENABLE	Resumes quota operations on a volume
EXIT	Returns the user to DCL command level
HELP	Lists the DISKQUOTA commands
MODIFY	Changes an entry in the quota file, and initializes the default quota and overdraft values
REBUILD	Reconstructs the usage counts for all entries
REMOVE	Deletes an entry from the quota file
SHOW	Displays quotas and usage counts
USE	Specifies the volume to be acted upon

DISK QUOTA
DISK QUOTA

FORMAT RUN SYS$SYSTEM:DISKQUOTA

PARAMETERS

None.

usage summary

To invoke the Disk Quota Utility, enter the following command at the DCL prompt:

```
$ RUN SYS$SYSTEM:DISKQUOTA
```

You can then enter DISKQUOTA commands at the DISKQ> prompt. These commands follow the standard rules of DCL syntax. To perform DISKQUOTA operations on a disk other than your current default disk (SYS$DISK), you must enter the USE command to specify the appropriate disk.

To exit from DISKQUOTA, enter the EXIT command at the DISKQ> prompt or press CTRL/Z. Either method returns control to the DCL command level.

No privileges are needed to invoke the Disk Quota Utility; however, most DISKQUOTA commands require write access to the quota file and many require the SYSPRV privilege, a system UIC, or ownership of the volume. See the individual command descriptions for details.

DISKQUOTA COMMANDS

This section describes the following DISKQUOTA commands and provides examples of their use:

ADD
CREATE
DISABLE
ENABLE
EXIT
HELP
MODIFY
REBUILD
REMOVE
SHOW
USE

ADD

Adds an entry to the quota file and initializes the usage count to zero.

This command requires write access to the quota file.

FORMAT

ADD *uic*

PARAMETER

uic

Specifies the user identification code (UIC) for which the quota entry is added. You can specify the UIC in numeric or alphanumeric format.

QUALIFIERS

/PERMQUOTA=quota

Specifies a positive integer that provides the quota for the specified UIC. If you omit the quota value, the quota defaults to the value of the quota in the entry for [0,0].

/OVERDRAFT=quota-plus

Specifies a positive integer that provides an overdraft value for the specified UIC. If you omit the quota-plus value, the overdraft value defaults to the quota-plus in the entry for [0,0].

DESCRIPTION

The ADD command appends individual entries to the quota file. Unless you specify the permanent quota and overdraft values, the utility applies the default values from the UIC entry [0,0]. (UIC [0,0] is the entry that you initialize immediately after creating a quota file, using the MODIFY command.)

EXAMPLE

```
DISKQ>ADD [MKT,MORSE] /PERMQUOTA=200/OVERDRAFT=50
```

The command in this example sets the permanent quota for UIC [MKT,MORSE] to 200 disk blocks and the overdraft limit to 50 disk blocks, for an absolute limit of 250 blocks.

CREATE

Creates a quota file for a volume that does not currently contain one.

This command requires write access to the volume's master file directory (MFD), as well as one of the following: the SYSPRV privilege, a system UIC, or ownership of the volume.

FORMAT

CREATE

PARAMETERS

None.

QUALIFIERS

None.

DESCRIPTION

The CREATE command creates and enables a quota file for a volume that does not currently have one.

Only one quota file, [000000]QUOTA.SYS, can be present on any volume or volume set. As soon as you create a quota file, you should establish default values for quotas and overdrafts by initializing UIC [0,0] with the MODIFY command. To have DISKQUOTA read a disk with existing files and build all UICs and their current usage into the newly-created quota file, use the REBUILD command.

EXAMPLE

```
DISKQ> CREATE
```

The command in this example creates a new quota file.

DISABLE

Suspends the maintenance and enforcement of quotas on a volume.

Use of this command requires one of the following: the SYSPRV privilege, a system UIC, or ownership of the volume.

FORMAT

DISABLE

PARAMETERS

None.

QUALIFIERS

None.

DESCRIPTION

The DISABLE command suspends the maintenance and enforcement of quotas. If you enable the quota file at a later point, enter the REBUILD command to update UIC entries and usage counts.

EXAMPLE

```
DISKQ> DISABLE
```

The command in this example suspends quota enforcement.

ENABLE

Resumes quota enforcement on a volume.

This command requires one of the following: the SYSPRV privilege, a system UIC, or ownership of the volume.

FORMAT

ENABLE

PARAMETERS

None.

QUALIFIERS

None.

DESCRIPTION

The ENABLE command lifts the suspension of quotas on a volume. Whenever you enable quotas on a volume, use the REBUILD command to update UIC entries and usage counts.

EXAMPLE

```
DISKQ> ENABLE
```

The command in this example resumes quota enforcement.

EXIT

Terminates the DISKQUOTA session and returns control to the DCL command level.

FORMAT

EXIT

PARAMETERS

None.

QUALIFIERS

None.

HELP

Lists and describes the DISKQUOTA commands.

FORMAT

HELP *[command]*

PARAMETER

command
Specifies the name of a DISKQUOTA command.

QUALIFIERS

None.

EXAMPLE

```
DISKQ> HELP ENABLE

  ENABLE

     Enable disk quotas on the default volume.  The default volume may
     be selected by issuing the USE command.  Once a quota file is created
     on a volume, disk quotas are assumed enabled until explicitly
     disabled.

     FORMAT:
      ENABLE
```

The command in this example displays information about the ENABLE command.

MODIFY

Changes an entry in the quota file or initializes default values for quotas and overdrafts. If a new quota limit is less than the current usage count, the utility issues a warning message before it implements the new quota.

The MODIFY command requires write access to the quota file.

FORMAT

MODIFY *uic*

PARAMETER

uic

Specifies the user identification code (UIC). You can specify the UIC in numeric or alphanumeric format, and the asterisk wildcard character (*) is permitted.

QUALIFIERS

/PERMQUOTA=quota

Specifies a positive integer that provides the quota for the specified UIC. If you omit the quota value, the quota defaults to the value of the quota in the entry for [0,0].

/OVERDRAFT=quota-plus

Specifies a positive integer that provides an overdraft value for the specified UIC. If you omit the quota-plus value, the overdraft value defaults to the quota-plus in the entry for [0,0].

DESCRIPTION

The MODIFY command changes values in the quota file. If you establish a quota limit that is less than the current usage count, a user can still log in and out, but cannot create files.

Use the MODIFY command after you have created a quota file to set appropriate default values for quotas and overdrafts. DISKQUOTA puts an entry into the newly-created quota file called UIC [0,0] that you initialize to values appropriate for your site.

EXAMPLE

```
DISKQ> MODIFY [0,0] /PERMQUOTA=3000 /OVERDRAFT=300
```

The command in this example sets the default permanent quota and overdraft values for the volume by editing the UIC entry [0,0]. If you do not specify a quota and overdraft, the utility applies these defaults to the UIC.

```
DISKQ> MODIFY [TTD,DAVIS] /PERMQUOTA=900
```

The command in this example sets the permanent quota for UIC [TTD,DAVIS] to 900 blocks, while making no change to the overdraft limit.

REBUILD

Adds entries to the quota file for files created while quotas were not being enforced, and updates the usage count for all UICs on the volume.

This command requires write access to the quota file, plus one of the following: the SYSPRV privilege, a system UIC, or ownership of the volume.

FORMAT

REBUILD

PARAMETERS

None.

QUALIFIERS

None.

DESCRIPTION

The REBUILD command reads the disk, updates usage counts for all existing entries, and adds new entries. It sets quota and overdraft values of new entries to the defaults set in UIC [0,0]. While the REBUILD command is executing, file activity on the volume is frozen. No files can be created, deleted, extended, or truncated. Use the REBUILD command in the following circumstances:

- After creating a quota file on a volume with existing files.
- When the quota file has been enabled after a period of being disabled. The command corrects the usage counts and adds any new UICs.

EXAMPLE

```
DISKQ> REBUILD
```

The command in this example reconstructs the usage counts for all entries on the volume.

REMOVE

Deletes an entry from the quota file.

This command requires write access to the quota file.

FORMAT

REMOVE *uic*

PARAMETER

uic

Specifies the user identification code (UIC). You can specify the UIC in numeric or alphanumeric format. The asterisk wildcard character (*) is permitted.

QUALIFIERS

None.

DESCRIPTION

If the usage count for the UIC is not zero, the utility issues a warning message before it removes the UIC. Files remain on disk, and the user can still log on; however, any attempt to create files will fail.

The UIC [0,0] entry cannot be removed.

EXAMPLE

```
DISKQ> REMOVE [TTD,DAVIS]
```

The command in this example deletes UIC [TTD,DAVIS] from the quota file.

SHOW

Displays quotas, overdrafts, and usage counts.

This command requires no privileges to show one's own quota, overdraft, and usage count, but otherwise it requires read access to the quota file.

FORMAT

SHOW *uic*

PARAMETER

uic

Specifies the user identification code (UIC). You can specify the UIC in numeric or alphanumeric format.

You can use an asterisk wildcard character (*) to specify the UIC as follows:

Command	Description
SHOW [TTD,CJ]	Show user CJ in group TTD
SHOW [TTD,*]	Show all users in group TTD
SHOW [*,CJ]	Show all users with a member name of CJ
SHOW [*]	Show all entries

QUALIFIERS

None.

EXAMPLE

```
DISKQ> SHOW [ACCT,*]
```

The command in this example displays quotas, overdrafts, and usage counts for all users in group ACCT.

USE

Specifies the volume to be acted upon.

FORMAT

USE *device*

PARAMETER

device
Specifies the physical or the logical name of the device for which disk quotas are being set, modified, or inspected.

QUALIFIERS

None.

DESCRIPTION

DISKQUOTA works with the default disk, SYS$DISK, unless the USE command is issued. Use this command during a DISKQUOTA session to switch from one quota file to another.

Any volume in a volume set can be specified.

EXAMPLE

```
DISKQ> USE DMA2:
```

The command in this example specifies the device by its physical name, DMA2:.

```
DISKQ> USE X2_RESEARCH_DATA
```

This command specifies the logical name of the physical device.

4.2 Obsolete Components of Current VMS Utilities

The following tables list obsolete components of currently supported utilities and the components that replace them.

Utility	Obsolete Component	Replacement
Runoff	.DO INDEX, .PRINT INDEX	RUNOFF/INDEX

The .DO INDEX and .PRINT INDEX commands, for in-core indexing, are no longer supported. RUNOFF/INDEX must be used to generate an .RNX file (index file) from a .BRN file. The .RNX file can then be .REQUIREd into the document.

Utility	Obsolete Component	Replacement
Librarian	$LBRCTLTBL	$LBRCTLDEF

In VAX/VMS Version 3.0 the macro $LBRCTLTBL in STARTLET.MLB defined certain librarian control table offsets. In VAX/VMS Version 4.0, the new structure definition translator renamed the macro $LBRCTLTBL to $LBRCTLDEF. If you have a program that references $LBRCTLTBL, remove the macro reference and reassemble your program. If for some reason the program does not run, then add the reference again using the new name, $LBRCTLDEF.

Utility	Obsolete Component	Replacement
Show Cluster	SHOW CLUSTER/REPORT=*x*	SHOW CLUSTER

The /REPORT=*x* qualifier is now unsupported. The information that was previously separated into two distinct reports, CLUSTER and LOCAL_PORTS, can now be displayed on the same screen by using the ADD command.

By default, the output of the SHOW CLUSTER command is the same as /REPORT=CLUSTER. To generate the LOCAL_PORTS report, (formerly done with the /REPORT=LOCAL_PORTS qualifier) create an initialization file containing the following commands:

```
REMOVE SYSTEMS, MEMBERS
ADD LOCAL_PORTS, ERRORS
```

A Eliminated Features

This appendix lists eliminated DCL commands, utilities, Run-Time Library (RTL) routines, and miscellaneous components of VMS. *Eliminated* means that the feature no longer works and has not been replaced.

Eliminated Features

The following table lists features eliminated from VMS:

DCL Commands	Eliminated Commands	Explanation
	SET PROCESS/CPU=[NO]ATTACHED	Support for the SET PROCESS /CPU=[NO]ATTACHED DCL command has been removed. This command was part of asymmetric multiprocessing (ASMP) support designed to help minimize scheduling inefficiencies. It has no counterpart under symmetric multiprocessing (SMP).
	SET COMMAND/NODELETE	The /NODELETE qualifier to the SET COMMAND command is no longer supported.
Utilities	**Eliminated Components**	**Explanation**
Command Definition	SET COMMAND/NODELETE	The /NODELETE qualifier to the SET COMMAND command is no longer supported.
Linker	/GSMATCH=NEVER	The NEVER keyword to the GSMATCH linker option is no longer allowed.
	/SHAREABLE=COPY	The COPY keyword on the options file qualifier /SHAREABLE is no longer allowed.
	UNIVERSAL=*	The linker option UNIVERSAL=* is no longer allowed.
SUMSLP	editor	The SLP editor is no longer supported.
RTL Routine	**Eliminated Routine**	**Explanation**
	SMG$ALLOW_ESCAPE	This routine was created solely for the purpose of translating old application programs that send escape sequences to SMG$, and is no longer supported.
Miscellaneous Features	**Eliminated Feature**	**Explanation**
Privileges	TMPJNL, PRMJNL	The TMPJNL and PRMJNL privileges were never used by VAX/VMS and have been removed.

Index

A

Accounting file
  controlling • 2–22
Accounting manager
  sending message to • 2–22
ADD command • 4–6

B

Batch job
  queue
    changing entry • 1–20

C

Channel
  information • 2–12
Control region
  deleting page from • 2–6
CPU (central processing unit)
  time
    to limit for batch job • 1–22
CREATE command • 4–7

D

Device
  information • 2–16
$DIBDEF macro
  symbol defined • 2–14
DISABLE command • 4–8

E

ENABLE command • 4–9
Equivalence name
  specifying • 2–8
EXIT command • 4–10

F

File
  to modify queue entry for • 1–20

H

HELP command • 4–11

I

INITIALIZE/QUEUE command
  /BURST qualifier • 1–29
  /[NO]FLAG qualifier • 1–30
  /PRIORITY qualifier • 1–30
  /TERMINAL qualifier • 1–30
Input request
  queuing and waiting for event flag • 2–19

L

Logical name
  creating • 2–8
  deleting • 2–10
  translating • 2–42

M

Message
  sending to accounting manager • 2–22
  writing to terminal • 2–2
MODIFY command • 4–12

O

Output
  queuing and waiting for event flag • 2–20

P

Print queue
  changing entry • 1–20
Program region
  deleting page from • 2–6

Q

Queue
  changing entry
    for batch • 1–20
    for printer • 1–20
  controlling print • 2–26
Quota file
  altering • 4–12
  deleting an entry • 4–14
  displaying an entry • 4–15

R

REBUILD command • 4–13
REMOVE command • 4–14

S

SET CLUSTER/QUORUM command • 1–2 to 1–3
SET DEVICE/ACL command • 1–4 to 1–7
SET DIRECTORY/ACL command • 1–8 to 1–13
SET FILE/ACL command • 1–14 to 1–19
SET QUEUE/ENTRY command • 1–20 to 1–28
SHOW command • 4–15
START/QUEUE command
  /BATCH qualifier • 1–30
  /PRIORITY qualifier • 1–31
  /TERMINAL qualifier • 1–31
Symbiont manager
  sending message to • 2–26
SYS$BRDCST • 2–2
  See also SYS$BRKTHRU
SYS$CNTREG • 2–6
  See also SYS$DELTVA
SYS$CRELOG • 2–8
  See also SYS$CRELNM
SYS$DELLOG • 2–10
  See also SYS$DELLNM
SYS$GETCHN • 2–12
  See also SYS$GETDVI
SYS$GETDEV • 2–16
  See also SYS$GETDVI
SYS$INPUT • 2–19
  See also SYS$QIO
SYS$OUTPUT • 2–20
  See also SYS$QIO
SYS$SNDACC • 2–22
  See also SYS$SNDJBC
SYS$SNDSMB • 2–26
  See also SYS$SNDJBC
SYS$TRNLOG • 2–42
  See also SYS$TRNLNM

U

Usage counts
  reconstructing • 4–13
USE command • 4–16

V

Virtual address space
  deleting page from • 2–6
Volume specification • 4–16

Reader's Comments

VMS Obsolete Features Manual
AA–LB25A–TE

Please use this postage-paid form to comment on this manual. If you require a written reply to a software problem and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Thank you for your assistance.

I rate this manual's:	Excellent	Good	Fair	Poor
Accuracy (software works as manual says)	☐	☐	☐	☐
Completeness (enough information)	☐	☐	☐	☐
Clarity (easy to understand)	☐	☐	☐	☐
Organization (structure of subject matter)	☐	☐	☐	☐
Figures (useful)	☐	☐	☐	☐
Examples (useful)	☐	☐	☐	☐
Index (ability to find topic)	☐	☐	☐	☐
Page layout (easy to find information)	☐	☐	☐	☐

I would like to see more/less _____

_____

_____

What I like best about this manual is _____

_____

_____

What I like least about this manual is _____

_____

_____

I found the following errors in this manual:

Page	Description
_____	_____
_____	_____
_____	_____
_____	_____
_____	_____

Additional comments or suggestions to improve this manual:

_____

_____

_____

_____

I am using **Version** _____ of the software this manual describes.

Name/Title _____ Dept. _____

Company _____ Date _____

Mailing Address _____

_____ Phone _____

— 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

DIGITAL EQUIPMENT CORPORATION
Corporate User Publications—Spit Brook
ZK01-3/J35 110 SPIT BROOK ROAD
NASHUA, NH 03062-9987

— Do Not Tear - Fold Here —

Cut Along Dotted Line

Reader's Comments

VMS Obsolete Features Manual
AA–LB25A–TE

Please use this postage-paid form to comment on this manual. If you require a written reply to a software problem and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Thank you for your assistance.

I rate this manual's:	Excellent	Good	Fair	Poor
Accuracy (software works as manual says)	☐	☐	☐	☐
Completeness (enough information)	☐	☐	☐	☐
Clarity (easy to understand)	☐	☐	☐	☐
Organization (structure of subject matter)	☐	☐	☐	☐
Figures (useful)	☐	☐	☐	☐
Examples (useful)	☐	☐	☐	☐
Index (ability to find topic)	☐	☐	☐	☐
Page layout (easy to find information)	☐	☐	☐	☐

I would like to see more/less _____

_____

_____

What I like best about this manual is _____

_____

_____

What I like least about this manual is _____

_____

_____

I found the following errors in this manual:

Page	Description
_____	_____
_____	_____
_____	_____
_____	_____
_____	_____

Additional comments or suggestions to improve this manual:

_____

_____

_____

_____

I am using **Version** _____ of the software this manual describes.

Name/Title _____ Dept. _____

Company _____ Date _____

Mailing Address _____

_____ Phone _____

-- 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

DIGITAL EQUIPMENT CORPORATION
Corporate User Publications—Spit Brook
ZK01-3/J35 110 SPIT BROOK ROAD
NASHUA, NH 03062-9987

-- Do Not Tear - Fold Here ----------

Cut Along Dotted Line