VMS

digital

VMS System Services Reference Manual

Order Number AA–LA69A–TE

VMS System Services Reference Manual

Order Number: AA–LA69A–TE

April 1988

This manual describes a set of routines the VMS operating system uses to control resources, to allow process communication, to control I/O, and to perform other such operating-system functions.

Revision/Update Information:	This manual supersedes the *VAX/VMS System Services Reference Manual,* Version 4.4.
Software Version:	VMS Version 5.0

digital equipment corporation
maynard, massachusetts

April 1988

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

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

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

Copyright ©1988 by Digital Equipment Corporation

All Rights Reserved.
Printed in U.S.A.

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™

ZK4527

HOW TO ORDER ADDITIONAL DOCUMENTATION
DIRECT MAIL ORDERS

USA & PUERTO RICO*

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 and Puerto Rico call 800-258-1710.

In New Hampshire, Alaska, and Hawaii call 603-884-6660.

In Canada call 800-267-6215.

*Any prepaid 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 trademark of Adobe Systems, Inc.

Contents

SYSTEM SERVICE DESCRIPTIONS

Contents

Preface

This manual provides reference information about the system services on the VMS operating system.

You can use VMS system services only in programs written in languages that produce native code for the VAX hardware. At present these languages include VAX MACRO and the following high-level languages:

VAX™ Ada®
VAX BASIC
VAX BLISS-32
VAX C
VAX COBOL
VAX COBOL-74
VAX CORAL
VAX DIBOL
VAX FORTRAN
VAX PASCAL
VAX PL/1

Intended Audience

This manual is intended for system and application programmers who want to call system services.

Document Structure

This manual provides detailed reference information about each system service. This information is presented using the documentation format described in the *Introduction to VMS System Services*. Service descriptions appear in alphabetical order by service name. Appendix A lists the obsolete services and the current services that have replaced them.

Readers should note that the introduction to the system services (formerly Part I) has been removed. For information and guidelines about using the system services, see the *Introduction to VMS System Services*.

Associated Documents

The *Introduction to VMS System Services* describes how to use the system services.

The VAX Procedure Calling and Condition Handling Standard, which is documented in the *Introduction to VMS System Routines*, contains useful information for anyone who wants to call system services.

VAX MACRO programmers can find additional information about calling system services in the *VAX MACRO and Instruction Set Reference Manual*.

™ VAX is a trademark of Digital Equipment Corporation.
® Ada is a registered trademark of the U.S. Government (Ada Joint Program Office).

High-level language programmers can find additional information about calling system services in the language reference manual and language user's guide provided with the VAX language.

The following documents may also be useful:

- *Guide to Using VMS Command Procedures*
- *Guide to VMS File Applications*
- *Guide to VMS System Security*
- *VMS Networking Manual*
- *VMS Record Management Services Manual*
- *VMS I/O User's Reference Manual: Part I*
- *VMS I/O User's Reference Manual: Part II*

For a complete list and description of the manuals in the VMS document set, see the *Overview of VMS Documentation.*

Conventions

The conventions used in this document are described in the *Introduction to VMS System Services.*

New and Changed Features

The section that follows describes the changes that have been made to the *VMS System Services Reference Manual* for VMS Version 5.0.

Modified Services

The following list describes the system services that have been modified for Version 5.0:

- The **flags** argument has been added to the following services:

  $SETIMR
  $SUSPND

- The $GETDVI service contains two new item codes.

  DVI$_DISPLAY_DEVNAM
  DVI$_TT_ACCPORNAM

- The following changes have been made to $GETLKI.

  The $GETLKI service contains new item codes.

  LKI$_CSID
  LKI$_CVTCOUNT
  LKI$_GRANTCOUNT
  LKI$_LKID
  LKI$_MSTCSID
  LKI$_MSTLKID
  LKI$_WAITCOUNT

  Four of the new item codes supersede existing item codes, which are supported for compatibility with VAX/VMS Version 4.n. DIGITAL recommends that you use the new item codes. You should update programs with the new item codes, as convenient.

  The following table lists the obsolete item codes and the new item codes that replace them:

Obsolete Item Code	New Item Code
LKI$_LCKCOUNT	LKI$_GRANTCOUNT
LKI$_REMLKID	LKI$_LKID LKI$_MSTLKID
LKI$_SYSTEM	LKI$_MSTCSID

  Four symbolic names have been changed for items returned by LKI$_BLOCKEDBY, LKI$_BLOCKING, and LKI$_LOCKS. The obsolete symbolic names are supported for compatibility with VAX/VMS Version 4.n. DIGITAL recommends that you use the new symbolic names. You should update programs with the new symbolic names, as convenient.

The following table lists the obsolete symbolic names and the new symbolic names that replace them.

Obsolete Symbolic Name	New Symbolic Name
LKI$L_LOCKID	LKI$L_MSTLKID
LKI$L_SYSID	LKI$L_MSTCSID
LKI$L_REMLKID	LKI$L_LKID
LKI$L_REMSYSID	LKI$L_CSID

- The $GETQUI service has the following additions:

  New function code

  QUI$_DISPLAY_ENTRY

  New item codes

  QUI$_EXECUTING_JOB_COUNT
  QUI$_FILE_IDENTIFICATION
  QUI$_HOLDING_JOB_COUNT
  QUI$_PENDING_JOB_BLOCK_COUNT
  QUI$_PENDING_JOB_COUNT
  QUI$_PENDING_JOB_REASON
  QUI$_QUEUE_DESCRIPTION
  QUI$_RESTART_QUEUE_NAME
  QUI$_RETAINED_JOB_COUNT
  QUI$_SEARCH_USERNAME
  QUI$_TIMED_RELEASE_JOB_COUNT

  New flags

  For the QUI$_JOB_STATUS item code

  QUI$V_JOB_PENDING
  QUI$V_JOB_SUSPENDED

  For the QUI$_QUEUE_FLAGS item code

  QUI$V_QUEUE_ACL_SPECIFIED
  QUI$V_QUEUE_PRINTER
  QUI$V_QUEUE_SERVER

  For the QUI$_QUEUE_STATUS item code

  QUI$V_QUEUE_CLOSED

  For the QUI$_SEARCH_FLAGS item code

  QUI$V_FREEZE_CONTEXT
  QUI$V_SEARCH_EXECUTING_JOBS
  QUI$V_SEARCH_GENERIC
  QUI$V_SEARCH_HOLDING_JOBS
  QUI$V_SEARCH_PENDING_JOBS
  QUI$V_SEARCH_PRINTER

QUI$V_SEARCH_RETAINED_JOBS
QUI$V_SEARCH_SERVER
QUI$V_SEARCH_TERMINAL
QUI$V_SEARCH_TIMED_RELEASE_JOBS

- The $GETSYI service contains the following new item codes:

  SYI$_ACTIVECPU_CNT
  SYI$_AVAILCPU_CNT
  SYI$_CLUSTER_ENODES
  SYI$_CONTIG_GBLPAGES
  SYI$_ERRORLOGBUFFERS
  SYI$_FREE_GBLPAGES
  SYI$_FREE_GBLSECTS
  SYI$_HW_MODEL
  SYI$_HW_NAME
  SYI$_NODE_EVOTES

  The SYI$_HW_NAME item code supersedes SYI$_NODE_HWTYPE, which is supported now for compatibility with VAX/VMS Version 4.n. DIGITAL recommends that you use SYI$_HW_NAME. You should update programs with the new item code, as convenient.

- The following changes have been made to $SNDJBC.

  The $SNDJBC service contains new item codes.

  SJC$_CLOSE_QUEUE
  SJC$_OPEN_QUEUE
  SJC$_PRINTER
  SJC$_QUEUE_DESCRIPTION, SJC$_NO_QUEUE_DESCRIPTION
  SJC$_SERVER

  The following item codes for $SNDJBC are supported now for compatibility with VAX/VMS Version 4.n, and may not be supported in the future.

  For the SJC$_CREATE_QUEUE function code

  SJC$_NO_TERMINAL

  For the SJC$_START_QUEUE function code

  SJC$_BATCH, SJC$_NO_BATCH
  SJC$_TERMINAL, SJC$_NO_TERMINAL

- The $MOUNT service contains the following new option for the MNT$_FLAGS item code:

  MNT$M_MULTI_VOL

- The ACL$C_JOBCTL_QUEUE object type has been added for the **objtyp** argument to the following services:

  $CHANGE_ACL
  $CHECK_ACCESS

- The $GETUAI and $SETUAI services have a new flag for the UAI$_FLAGS item code.

  UAI$V_FORCE_EXP_PWD_CHANGE

System Service Descriptions

$ADD_HOLDER Add Holder Record to Rights Database

The Add Holder Record to Rights Database service adds a specified holder record to a target identifier.

FORMAT

SYS$ADD_HOLDER *id ,holder ,[attrib]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

id

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

Target identifier granted to the specified holder when $ADD_HOLDER completes execution. The **id** argument is a longword containing the binary value of the target identifier.

holder

VMS usage: **rights_holder**
type: **quadword (unsigned)**
access: **read only**
mechanism: **by reference**

Holder identifier that is granted access to the target identifier when $ADD_HOLDER completes execution. The **holder** argument is the address of a quadword data structure that consists of a longword containing the holder's UIC identifier followed by a longword containing a value of zero.

attrib

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

Attributes to be placed in the holder record when the $ADD_HOLDER completes execution. The **attrib** argument is a longword containing a bit mask specifying the attributes. A holder is granted a specified attribute only if the target identifier has the attribute.

Symbol values are offsets to the bits within the longword. You can also obtain the values as masks with the appropriate bit set using the prefix KGB$M rather than KGB$V. The symbols are defined in the system macro library ($KGBDEF). The symbolic name for each bit position is listed in the following table:

Bit Position	Meaning When Set
KGB$V_DYNAMIC	Allows the unprivileged holder to add or remove the identifier from the process rights list
KGB$V_RESOURCE	Allows the holder to charge resources, such as disk blocks, to the identifier

DESCRIPTION

The Add Holder Record to Rights Database service registers the specified user as a holder of the specified identifier with the rights database.

You need write access to the rights database to use this service. If the database is in SYS$SYSTEM, which is the default, you need SYSPRV privilege to grant write access to the database.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The **holder** argument cannot be read by the caller.
SS$_BADPARAM	The specified attributes contain invalid attribute flags.
SS$_DUPIDENT	The specified holder already exists in the rights database for this identifier.
SS$_INSFMEM	The process dynamic memory is insufficient for opening the rights database.
SS$_IVIDENT	The specified identifier or holder is of invalid format, or the specified identifier and holder are equal.
SS$_NOSUCHID	The specified identifier does not exist in the rights database, or the specified holder identifier does not exist in the rights database.
RMS$_PRV	The user does not have write access to the rights database.

Because the rights database is an indexed file accessed with VMS RMS, this service may also return RMS status codes associated with operations on indexed files. For descriptions of these status codes, refer to the *VMS Record Management Services Manual.*

$ADD_IDENT Add Identifier to Rights Database

The Add Identifier to Rights Database service adds the specified identifier to the rights database.

FORMAT

SYS$ADD_IDENT *name ,[id] ,[attrib] ,[resid]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

name

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

Identifier name to be added to the rights database when $ADD_IDENT completes execution. The **name** argument is the address of the descriptor pointing to the identifier name string.

An identifier name consists of 1 to 31 alphanumeric characters, including dollar signs ($) and underscores (_), and must contain at least one nonnumeric character. Any lowercase characters specified are automatically converted to uppercase.

id

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

Identifier to be created when $ADD_IDENT completes execution. The **id** argument is a longword containing the binary value of the identifier to be created.

If omitted, $ADD_IDENT selects a unique available value from the general identifier space and returns it in **resid**, if it is specified.

attrib

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

Attributes placed in the identifier's record when $ADD_IDENT completes execution. The **attrib** argument is a longword containing a bit mask that specifies the attributes.

Symbol values are offsets to the bits within the longword. You can also obtain the values as masks with the appropriate bit set using the prefix KGB$M rather than KGB$V. The symbols are defined in the system macro library ($KGBDEF). The symbolic name for each bit position is listed in the following table:

Bit Position	Meaning When Set
KGB$V_DYNAMIC	Allows the unprivileged holder to add or remove the identifier from the process rights list
KGB$V_RESOURCE	Allows the holder to charge resources, such as disk blocks, to the identifier

resid

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

Identifier value assigned by the system when $ADD_IDENT completes execution. The **resid** argument is the address of a longword in which the system-assigned identifier value is written.

DESCRIPTION

The Add Identifier to Rights Database service adds the specified identifier to the rights database.

You need write access to the rights database to use this service. If the database is in SYS$SYSTEM, which is the default, you need SYSPRV privilege to grant write access to the database.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The **name** argument cannot be read by the caller, or the **resid** argument cannot be written by the caller.
SS$_BADPARAM	The specified attributes contain invalid attribute flags.
SS$_DUPIDENT	The specified identifier already exists in the rights database.
SS$_DUPLNAM	The specified identifier name already exists in the rights database.
SS$_INSFMEM	The process dynamic memory is insufficient for opening the rights database.
SS$_IVIDENT	The specified identifier is of invalid format.
RMS$_PRV	The user does not have write access to the rights database.

Because the rights database is an indexed file accessed with VMS RMS, this service may also return RMS status codes associated with operations on indexed files. For descriptions of these status codes, refer to the *VMS Record Management Services Manual.*

$ADJSTK Adjust Outer Mode Stack Pointer

The Adjust Outer Mode Stack Pointer service modifies the stack pointer for a less privileged access mode. The operating system uses this service to modify a stack pointer for a less privileged access mode after placing arguments on the stack.

FORMAT

SYS$ADJSTK *[acmode],[adjust],newadr*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

acmode

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

Access mode for which the stack pointer is to be adjusted. The **acmode** argument is this longword value. If not specified, the default value 0 (kernel access mode) is used.

adjust

VMS usage: **word_signed**
type: **word (signed)**
access: **read only**
mechanism: **by value**

Signed adjustment value used to modify the value specified by the **newadr** argument. The **adjust** argument is a signed longword, which is the adjustment value.

Only the low-order word of this argument is used. The value specified by the low-order word is added or subtracted (depending on the sign) from the value specified by the **newadr** argument. The result is loaded into the stack pointer for the specified access mode.

If the **adjust** argument is not specified or is specified as 0, the stack pointer is loaded with the value specified by the **newadr** argument.

For additional information about the various combinations of values for **adjust** and **newadr**, see the Description section.

newadr

VMS usage: **address**
type: **longword (unsigned)**
access: **modify**
mechanism: **by reference**

Value that $ADJUST is to adjust. The **newadr** argument is the address of this longword value. The value specified by this argument is both read and written by $ADJSTK. The $ADJSTK service reads the value specified and adjusts it by the value of the **adjust** argument (if specified). After this adjustment is made, $ADJSTK writes the adjusted value back into the longword specified by **newadr** and then loads the stack pointer with the adjusted value.

If the value specified by **newadr** is 0, the current value of the stack pointer is adjusted by the value specified by **adjust**. This new value is then written back into **newadr**, and the stack pointer is modified.

For additional information about the various combinations of values for **adjust** and **newadr**, see the Description section.

DESCRIPTION

Combinations of zero and nonzero values for the **adjust** and **newadr** arguments provide the following results:

If the adjust argument specifies:	And the value specified by newadr is:	The stack pointer is:
0	0	Not changed
0	An address	Loaded with the address specified
A value	0	Adjusted by the specified value
A value	An address	Loaded with the specified address, adjusted by the specified value

In all cases, the updated stack pointer value is written into the value specified by the **newadr** argument.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The value specified by **newadr** or a portion of the new stack segment cannot be written by the caller.
SS$_NOPRIV	The specified access mode is equal to or more privileged than the calling access mode.

$ADJWSL Adjust Working Set Limit

The Adjust Working Set Limit service adjusts a process's current working set limit by the specified number of pages and returns the new value to the caller. The working set limit specifies the maximum number of process pages that can be resident in physical memory.

FORMAT

SYS$ADJWSL *[pagcnt],[wsetlm]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

pagcnt

VMS usage: **longword_signed**
type: **longword (signed)**
access: **read only**
mechanism: **by value**

Signed adjustment value specifying the number of pages to add to (if positive) or subtract from (if negative) the current working set limit. The **pagcnt** argument is this signed longword value.

If **pagcnt** is not specified or is specified as 0, no adjustment is made and the current working set limit is returned in the longword specified by the **wsetlm** argument (if this argument is specified).

wsetlm

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

Value of the working set limit, returned by $ADJWSL. The **wsetlm** argument is the address of this longword value. The **wsetlm** argument specifies the newly adjusted value if **pagcnt** is specified, and it specifies the old, unadjusted value if **pagcnt** is not specified.

DESCRIPTION

If a program attempts to adjust the working set limit beyond the system-defined upper and lower limits, no error condition is returned; instead, the working set limit is adjusted to the maximum or minimum size allowed.

The initial value of a process's working set limit is controlled by the working set default (WSDEFAULT) quota. The maximum value to which it may be increased is controlled by the working set extent (WSEXTENT) quota; the minimum value to which it may be decreased is limited by the SYSGEN parameter MINWSCNT.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The longword specified by **wsetlm** cannot be written by the caller.

$ALLOC Allocate Device

The Allocate Device service allocates a device for exclusive use by a process and its subprocesses. No other process can allocate the device or assign channels to it until the image that called $ALLOC exits or explicitly deallocates the device with the Deallocate Device ($DALLOC) service.

FORMAT

SYS$ALLOC *devnam ,[phylen] ,[phybuf] ,[acmode] ,[flags]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns 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**

Device name of the device to be allocated. The **devnam** argument is the address of a character string descriptor pointing to the device name string.

The string may be either a physical device name or a logical name. If it is a logical name, it must translate to a physical device name.

phylen

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

Word into which $ALLOC writes the length of the device name string for the device it has allocated. The **phylen** argument is the address of this word.

phybuf

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

Buffer into which $ALLOC writes the device name string for the device it has allocated. The **phybuf** argument is the address of a character string descriptor pointing to this buffer.

acmode

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

Access mode to be associated with the allocated device. The **acmode** argument is a longword containing the access mode.

The most privileged access mode used is the access mode of the caller. Only equal or more privileged access modes can deallocate the device.

flags

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

Longword of status flags indicating whether to interpret the **devnam** argument as the type of device to be allocated. Only one flag exists, bit 0. When it is set, the $ALLOC service allocates the first available device that has the type specified in the **devnam** argument.

This feature is available for the following mass storage devices:

RA60	RA80	RA81	RC25
RCF25	RK06	RK07	RL01
RL02	RM03	RM05	RM80
RP04	RP05	RP06	RP07
RX01	RX02	TA78	TA81
TS11	TU16	TU58	TU77
TU78	TU80	TU81	

DESCRIPTION

The calling process must have ALLSPOOL privilege to allocate a spooled device.

When a process calls the Assign I/O Channel ($ASSIGN) service to assign a channel to a nonshareable, nonspooled device, such as a terminal or line printer, the device is implicitly allocated to the process.

You can use this service only to allocate devices that either exist on the host system or are made available to the host system in a VAXcluster environment.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_BUFFEROVF	The service completed successfully. The physical name returned overflowed the buffer provided, and has been truncated.
SS$_DEVALRALLOC	The service completed successfully. The device was already allocated to the calling process.

SS$_ACCVIO	The device name string, string descriptor, or physical name buffer descriptor cannot be read by the caller; or the physical name buffer cannot be written by the caller.
SS$_DEVALLOC	The device is already allocated to another process, or an attempt to allocate an unmounted shareable device failed because other processes had channels assigned to the device.
SS$_DEVMOUNT	The specified device is currently mounted and cannot be allocated, or the device is a mailbox.
SS$_DEVOFFLINE	The specified device is marked off line.
SS$_IVDEVNAM	The device name string contains invalid characters, or no device name string was specified.
SS$_IVLOGNAM	The device name string has a length of 0 or has more than 63 characters.
SS$_IVSTSFLG	The bits set in the longword of status flags are invalid.
SS$_NODEVAVL	The specified device in a generic search exists but is allocated to another user.
SS$_NONLOCAL	The device is on a remote node.
SS$_NOPRIV	The requesting process attempted to allocate a spooled device, and does not have the required privilege; or the device protection or access control list (or both) denies access.
SS$_NOSUCHDEV	The specified device does not exist in the host system. This error is usually the result of a typographical error.
SS$_TEMPLATEDEV	The process attempted to allocate a template device; a template device cannot be allocated.

The $ALLOC service can also return any condition value returned by $ENQ. For a list of these condition values, see the description of $ENQ.

$ASCEFC Associate Common Event Flag Cluster

The Associate Common Event Flag Cluster service causes a named common event flag cluster to be associated with a process for the execution of the current image and to be assigned a process-local cluster number for use with other event flag services. If the named cluster does not exist but the process has suitable privilege, the service creates the cluster.

FORMAT

SYS$ASCEFC *efn ,name ,[prot] ,[perm]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

efn

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

Number of any event flag contained within the desired common event flag cluster. The **efn** argument is a longword value specifying this number; however, $ASCEFC uses only the low-order byte.

There are two common event flag clusters: cluster 2 and cluster 3. Cluster 2 contains event flag numbers 64 to 95, and cluster 3 contains event flag numbers 96 to 127. (Clusters 0 and 1 are process-local event flag clusters.)

To associate with common event flag cluster 2, specify any flag number in the cluster (64 to 95); to associate with common event flag cluster 3, specify any event flag number in the cluster (96 to 127).

name

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

Name of the common event flag cluster with which to associate. The **name** argument is the address of a character string descriptor pointing to this name string.

Common event flag clusters are accessible only to processes having the same UIC group number, and each such process must associate with the cluster using the same name (specified in the **name** argument). VMS implicitly

associates the group UIC number with the name, making the name unique to a UIC group.

prot

VMS usage: **boolean**
type: **byte (unsigned)**
access: **read only**
mechanism: **by value**

Protection specifier that allows or disallows access to the common event flag cluster for processes with the same UIC group number as the creating process. The **prot** argument is a longword value, which is interpreted as Boolean.

The default value 0 specifies that any process with the same UIC group number as the creator may access the event flag cluster. The value 1 specifies that only processes with the creator's UIC can access the event flag cluster.

perm

VMS usage: **boolean**
type: **byte (unsigned)**
access: **read only**
mechanism: **by value**

Permanent specifier that marks a common event flag cluster as either permanent or temporary. The **perm** argument is a longword value, which is interpreted as Boolean.

The default value 0 specifies that the cluster is temporary. The value 1 specifies that the cluster is permanent.

DESCRIPTION

The calling process must have PRMCEB privilege to create a permanent common event flag cluster.

Creation of temporary common event flag clusters uses the quota of the process for timer queue entries (TQELM); the creation of a permanent cluster does not affect the quota. The quota is restored to the creator of the cluster when all processes associated with the cluster have disassociated.

When a process associates with a common event flag cluster, that cluster's reference count is increased by 1. The reference count is decreased when a process disassociates from the cluster, whether explicitly with the Disassociate Common Event Flag Cluster ($DACEFC) service or implicitly at image exit.

Temporary clusters are automatically deleted when their reference count goes to 0; you must explicitly mark permanent clusters for deletion with the Delete Common Event Flag Cluster ($DLCEFC) service.

Because the $ASCEFC service automatically creates the common event flag cluster if it does not already exist, cooperating processes need not be concerned with which process executes first to create the cluster. The first process to call $ASCEFC creates the cluster and the others associate with it regardless of the order in which they call the service.

The initial state for all event flags in a newly created common event flag cluster is 0.

If a process has already associated a cluster number with a named common event flag cluster and then issues another call to $ASCEFC with the same cluster number, the service disassociates the number from its first assignment before associating it with its second.

If you previously called any system service that will set an event flag (and the event flag is contained within the cluster being reassigned), the event flag will be set in the newly associated named cluster, not the previously associated named cluster.

For more information about common event flag clusters in shared memory, refer to *Introduction to VMS System Services.*

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The cluster name string or string descriptor cannot be read by the caller.
SS$_EXPORTQUOTA	The process has exceeded the number of event flag clusters with which processes on this port of the multiport (shared) memory can associate.
SS$_EXQUOTA	The process has exceeded its timer queue entry quota; this quota controls the creation of temporary common event flag clusters.
SS$_INSFMEM	The system dynamic memory is insufficient for completing the service.
SS$_ILLEFC	You specified an illegal event flag number. The cluster number must be in the range of event flags 64 through 127.
SS$_INTERLOCK	The bit map lock for allocating common event flag clusters from the specified shared memory is locked by another process.
SS$_IVLOGNAM	The cluster name string has a length of 0 or has more than 15 characters.
SS$_NOPRIV	The process does not have the privilege to create a permanent cluster; the process does not have the privilege to create a common event flag cluster in memory shared by multiple processors, or the protection applied to an existing cluster by its creator prohibits association.
SS$_NOSHMBLOCK	The common event flag cluster has no shared memory control block available.
SS$_SHMNOTCNCT	The shared memory named in the **name** argument is not known to the system. This error can be caused by a spelling error in the string, an improperly assigned logical name, or the failure to identify the memory as shared at system generation time.

$ASCTIM Convert Binary Time to ASCII String

The Convert Binary Time to ASCII String service converts an absolute or delta time from 64-bit system time format to an ASCII string.

FORMAT

SYS$ASCTIM *[timlen] ,timbuf ,[timadr] ,[cvtflg]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

timlen

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

Length (in bytes) of the ASCII string returned by $ASCTIM. The **timlen** argument is the address of a word containing this length.

timbuf

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

Buffer into which $ASCTIM writes the ASCII string. The **timbuf** argument is the address of a character string descriptor pointing to the buffer.

The buffer length specified in the **timbuf** argument, together with the **cvtflg** argument, controls what information is returned.

timadr

VMS usage: **date_time**
type: **quadword (unsigned)**
access: **read only**
mechanism: **by reference**

Time value that $ASCTIM is to convert. The **timadr** argument is the address of this 64-bit time value. A positive time value represents an absolute time. A negative time value represents a delta time. If you specify a delta time, it must be less than 10,000 days.

If **timadr** is not specified or is specified as 0 (the default), $ASCTIM returns the current date and time.

cvtflg

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

Conversion indicator specifying which date and time fields $ASCTIM should return. The **cvtflg** argument is a longword value, which is interpreted as Boolean. The value 1 specifies that $ASCTIM should return only the hour, minute, second, and hundredths of second fields. The default value 0 specifies that $ASCTIM should return the full date and time.

DESCRIPTION

The $ASCTIM service executes at the access mode of the caller and does not check whether address arguments are accessible before it executes. Therefore, an access violation causes an exception condition if the input time value cannot be read or the output buffer or buffer length cannot be written.

This service does not check the length of the argument list, and therefore cannot return the SS$_INSFARG (insufficient arguments) condition value.

The ASCII strings returned have the following formats:

Absolute Time: dd-mmm-yyyy hh:mm:ss.cc

Delta Time: dddd hh:mm:ss.cc

The following table lists the length (bytes), contents, and range of values for each field in the absolute time and delta time formats.

Field	Length (Bytes)	Contents	Range of Values
dd	2	Day of month	1–31
–	1	Hyphen	Required syntax
mmm	3	Month	JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC
–	1	Hyphen	Required syntax
yyyy	4	Year	1858–9999
blank	n	Blank	Required syntax
hh	2	Hour	00–23
:	1	Colon	Required syntax
mm	2	Minutes	00–59
:	1	Colon	Required syntax
ss	2	Seconds	00–59
.	1	Period	Required syntax
cc	2	Hundredths of second	00–99
dddd	4	Number of days (in 24-hr units)	000–9999

Month abbreviations must be uppercase. The hundredths of second field now represents a true fraction; for example, the string .1 represents ten hundredths of a second (one tenth of a second); the string .01 represents one hundredth of a second.

Also, you can add a third digit to the hundredths of second field; this thousandths of second digit is used to round the hundredths of second value. Digits beyond the thousandths of second digits are ignored.

The results of specifying some possible combinations for the values of the **cvtflg** and **timbuf** arguments are as follows:

Time Value	Buffer Length Specified	CVTFLG Argument	Information Returned
Absolute	23	0	Date and time
Absolute	12	0	Date
Absolute	11	1	Time
Delta	16	0	Days and time
Delta	11	1	Time

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_BUFFEROVF	The buffer length specified in the **timbuf** argument is too small.
SS$_IVTIME	The specified delta time is equal to or greater than 10,000 days.

$ASCTOID Translate Identifier Name to Identifier

The Translate Identifier Name to Identifier service translates the specified identifier name into its binary identifier value.

FORMAT

SYS$ASCTOID *name ,[id] ,[attrib]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

name

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

Identifier name translated when $ASCTOID completes execution. The **name** argument is the address of a descriptor pointing to the identifier name.

id

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

Identifier value resulting when $ASCTOID completes execution. The **id** argument is the address of a longword in which the identifier value is written.

attrib

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

Attributes associated with the identifier returned in **id** when $ASCTOID completes execution. The **attrib** argument is the address of a longword containing a bit mask specifying the attributes.

Symbol values are offsets to the bits within the longword. You can also obtain the values as masks with the appropriate bit set using the prefix KGB$M rather than KGB$V. The symbols are defined in the system macro library ($KGBDEF). The symbolic names for each bit position are listed in the following table:

Bit Position	Meaning When Set
KGB$V_DYNAMIC	Allows the unprivileged holder to add or remove the identifier from the process rights list
KGB$V_RESOURCE	Allows the holder to charge resources, such as disk blocks, to the identifier

DESCRIPTION

The Translate Identifier Name to Identifier converts the specified identifier name to its binary identifier value. Note that when you use wildcards with this service, the records are returned alphabetically by identifier name.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The name argument cannot be read by the caller, or the **id** or **attribute** argument cannot be written by the caller.
SS$_INSFMEM	The process dynamic memory is insufficient for opening the rights database.
SS$_IVIDENT	The specified identifier is of invalid format.
SS$_NOSUCHID	The specified identifier name does not exist in the rights database.
RMS$_PRV	The user does not have read access to the rights database.

Because the rights database is an indexed file accessed with VMS RMS, this service may also return RMS status codes associated with operations on indexed files. For descriptions of these status codes, refer to the *VMS Record Management Services Manual.*

$ASSIGN Assign I/O Channel

The Assign I/O Channel service (1) provides a process with an I/O channel so that input/output operations can be performed on a device or (2) establishes a logical link with a remote node on a network.

FORMAT

SYS$ASSIGN *devnam ,chan ,[acmode] ,[mbxnam]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns 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 to which $ASSIGN is to assign a channel. The **devnam** argument is the address of a character string descriptor pointing to the device name string.

If the device name contains a double colon (::), the system assigns a channel to the first available network device (NET:) and performs an access function on the network.

chan

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

Number of the channel that is assigned. The **chan** argument is the address of a word into which $ASSIGN writes the channel number.

acmode

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

Access mode to be associated with the channel. The **acmode** argument specifies the access mode. The most privileged access mode used is the access mode of the caller. I/O operations on the channel can be performed only from equal and more privileged access modes.

mbxnam

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

Logical name of the mailbox to be associated with the device. The **mbxnam** argument is the address of a character string descriptor pointing to the logical name string.

If you specify **mbxnam** as 0, no mailbox is associated with the device. This is the default.

You must specify the **mbxnam** argument when performing a nontransparent task-to-task DECnet–VAX operation.

Only the owner of a device can associate a mailbox with the device; the owner of a device is the process that has allocated the device, whether implicitly or explicitly. Only one mailbox can be associated with a device at any one time.

A mailbox cannot be associated with a device if the device has foreign (DEV$M_FOR) or shareable (DEV$M_SHR) characteristics.

A mailbox is disassociated from a device when the channel that associated it is deassigned.

If a mailbox is associated with a device, the device driver can send status information to the mailbox. For example, if the device is a terminal, this information may indicate dial-up, hang-up, or the reception of unsolicited input; if the device is a network device, it may indicate that the network is connected or perhaps that the line is down.

For details on the nature and format of the information returned to the mailbox, refer to the *VMS I/O User's Reference Volume.*

DESCRIPTION

The calling process must have NETMBX privilege to perform network operations.

System dynamic memory is required if the target device is on a remote system.

Channels remain assigned until they are explicitly deassigned with the Deassign I/O Channel ($DASSGN) service or, if they are user-mode channels, until the image that assigned the channel exits.

The $ASSIGN service establishes a path to a device but does not check whether the caller can actually perform input/output operations to the device. Privilege and protection restrictions may be applied by the device drivers.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_REMOTE	The service completed successfully. A logical link is established with the target on a remote node.

SS$_ABORT	A physical line went down during a network connect operation.
SS$_ACCVIO	The device or mailbox name string or string descriptor cannot be read by the caller, or the channel number cannot be written by the caller.
SS$_DEVACTIVE	You specified a mailbox name, but a mailbox is already associated with the device.
SS$_DEVALLOC	The device is allocated to another process.
SS$_DEVNOTMBX	You specified a logical name for the associated mailbox, but the logical name refers to a device that is not a mailbox.
SS$_EXQUOTA	The target of the assignment is on a remote node and the process has insufficient buffer quota to allocate a network control block.
SS$_INSFMEM	The target of the assignment is on a remote node and there is insufficient system dynamic memory to complete the request.
SS$_IVDEVNAM	No device name was specified, the logical name translation failed, or the device or mailbox name string contains invalid characters. If the device name is a target on a remote node, this status code indicates that the Network Connect Block has an invalid format.
SS$_IVLOGNAM	The device or mailbox name string has a length of 0 or has more than 63 characters.
SS$_NOIOCHAN	No I/O channel is available for assignment.
SS$_NOLINKS	For network operations, no logical links are available. The maximum number of logical links as set for the NCP executor MAXIMUM LINKS parameter was exceeded.
SS$_NOPRIV	For network operations, the issuing task does not have the required privilege to perform network operations or to confirm the specified logical link.
SS$_NOSUCHDEV	The specified device or mailbox does not exist, or, for DECnet–VAX operations, the network device driver is not loaded (for example, the DECnet–VAX software is not currently running on the local VAX node).
SS$_NOSUCHNODE	The specified network node is nonexistent or unavailable.
SS$_REJECT	The network connect was rejected by the network software or by the partner at the remote node, or the target image exited before the connect confirm could be issued.
SS$_CONNECFAIL	For network operations, the connection to a network object timed out or failed.
SS$_DEVOFFLINE	For network operations, the physical link is shutting down.

SS$_FILALRACC	For network operations, a logical link already exists on the channel.
SS$_INVLOGIN	For network operations, the access control information was found to be invalid at the remote node.
SS$_LINKEXIT	For network operations, the network partner task was started, but exited before confirming the logical link (that is, $ASSIGN to SYS$NET).
SS$_NOSUCHOBJ	For network operations, the network object number is unknown at the remote node; for a TASK= connect, the named DCL command procedure file cannot be found at the remote node.
SS$_NOSUCHUSER	For network operations, the remote node could not recognize the login information supplied with the connection request.
SS$_PROTOCOL	For network operations, a network protocol error occurred, most likely because of a network software error.
SS$_REMRSRC	For network operations, the link could not be established because system resources at the remote node were insufficient.
SS$_SHUT	For network operations, the local or remote node is no longer accepting connections.
SS$_THIRDPARTY	For network operations, the logical link connection was terminated by a third party (for example, the system manager).
SS$_TOOMUCHDATA	For network operations, the task specified too much optional or interrupt data.
SS$_UNREACHABLE	For network operations, the remote node is currently unreachable.

$BINTIM Convert ASCII String to Binary Time

The Convert ASCII String to Binary Time service converts an ASCII string to an absolute or delta time value in the system 64-bit time format suitable for input to the Set Timer ($SETIMR) or Schedule Wakeup ($SCHDWK) service.

FORMAT

SYS$BINTIM *timbuf,timadr*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

timbuf

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

Buffer that holds the ASCII time to be converted. The **timbuf** argument specifies the address of a character string descriptor pointing to the VMS time string. The VMS time string specifies the absolute or delta time to be converted by $BINTIM. The VMS Data Type Table describes the VMS time string.

timadr

VMS usage: **date_time**
type: **quadword (unsigned)**
access: **write only**
mechanism: **by reference**

Time value that $BINTIM has converted. The **timadr** argument is the address of the VMS quadword system time, which receives the converted time.

DESCRIPTION

The $BINTIM service executes at the access mode of the caller and does not check whether address arguments are accessible before it executes. Therefore, an access violation causes an exception condition if the input buffer or buffer descriptor cannot be read or the output buffer cannot be written.

This service does not check the length of the argument list and therefore cannot return the SS$_INSFARG (insufficient arguments) error status code. If the service does not receive enough arguments (for example, if you omit required commas in the call), errors may result.

The required ASCII input strings have the following format:

Absolute Time: dd-mmm-yyyy hh:mm:ss.cc

Delta Time: dddd hh:mm:ss.cc

The following table lists the length (bytes), contents, and range of values for each field in the absolute time and delta time formats.

Field	Length (Bytes)	Contents	Range of Values
dd	2	Day of month	1–31
–	1	Hyphen	Required syntax
mmm	3	Month	JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC
–	1	Hyphen	Required syntax
yyyy	4	Year	1858–9999
blank	n	Blank	Required syntax
hh	2	Hour	00–23
:	1	Colon	Required syntax
mm	2	Minutes	00–59
:	1	Colon	Required syntax
ss	2	Seconds	00–59
.	1	Period	Required syntax
cc	2	Hundredths of second	00–99
dddd	4	Number of days (in 24-hour units)	000–9999

Note that month abbreviations must be uppercase and that the hundredths of second field represents a true fraction. For example, the string .1 represents ten hundredths of a second (one tenth of a second) and the string .01 represents one hundredth of a second. Note also that you can add a third digit to the hundredths of second field; this thousandths of second digit is used to round the hundredths of second value. Digits beyond the thousandths of second digits are ignored.

The following two syntax rules apply to specifying the ASCII input string:

- You can omit any of the date and time fields.

  For absolute time values, the $BINTIM service supplies the current system date and time for nonspecified fields. Trailing fields can be truncated. If leading fields are omitted, you must specify the punctuation (hyphens, blanks, colons, periods). For example, the following string results in an absolute time of 12:00 on the current day:

```
-- 12:00:00.00
```

For delta time values, the $BINTIM service uses a default value of 0 for unspecified hours, minutes, and seconds fields. Trailing fields can be truncated. If you omit leading fields from the time value, you must specify the punctuation (blanks, colons, periods). If the number of days in the delta time is 0, you must specify a 0. For example, the following string results in a delta time of 10 seconds:

```
0 ::10
```

Note the space between the 0 in the day field and the two colons.

- For both absolute and delta time values, there can be any number of leading blanks, and any number of blanks between fields normally delimited by blanks. However, there can be no embedded blanks within either the date or time field.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_IVTIME	The syntax of the specified ASCII string is invalid, or the time component is out of range.

EXAMPLE

Column 1 of the following table lists legal input strings to the $BINTIM service; column 2 lists the $BINTIM output of these strings translated through the Convert Binary Time to ASCII String ($ASCTIM) system service. The current date is assumed to be 30-DEC-1988 04:15:28.00.

Input to $BINTIM	$ASCTIM Output String
-- :50	30-DEC-1988 04:50:28.00
--1989 0:0:0.0	29-DEC-1989 00:00:00.00
30-DEC-1988 12:32:1.1161	30-DEC-1988 12:32:01.12
29-DEC-1989 16:35:0.0	29-DEC-1989 16:35:00.00
0 ::.1	0 00:00:00.10
0 ::.06	0 00:00:00.06
5 3:18:32.068	5 03:18:32:07
20 12:	20 12:00:00.00
0 5	0 05:00:00.00

$BRKTHRU Breakthrough

The Breakthrough service sends a message to one or more terminals. The $BRKTHRU service completes asynchronously; that is, it returns to the caller after queueing the message request, without waiting for the message to be written to the specified terminal(s).

For synchronous completion, use the Breakthrough and Wait ($BRKTHRUW) service. The $BRKTHRUW service is identical to the $BRKTHRU service in every way except that $BRKTHRUW returns to the caller after the message is written to the specified terminal(s).

For additional information about system service completion, refer to the Synchronize ($SYNCH) service and to the *Introduction to VMS System Services*.

The $BRKTHRU service supersedes the Broadcast ($BRDCST) service. When writing new programs, you should use $BRKTHRU instead of $BRDCST. When updating old programs, you should change all uses of $BRDCST to $BRKTHRU.

FORMAT

SYS$BRKTHRU *[efn],msgbuf[,sendto][,sndtyp][,iosb][,carcon][,flags][,reqid][,timout][,astadr][,astprm]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

efn

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

Number of the event flag to be set when the message has been written to the specified terminal(s). The **efn** argument is a longword containing this number; however, $BRKTHRU uses only the low-order byte.

When the message request is queued, $BRKTHRU clears the specified event flag (or event flag 0 if **efn** is not specified). Then, after the message is sent, $BRKTHRU sets the specified event flag (or event flag 0).

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 $BRKTHRU service permits the message text to be as long as 16,350 bytes; however, both the SYSGEN parameter MAXBUF and the caller's available process space may affect the maximum length of the message text.

sendto

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

Name of a single device (terminal) or single user name to which the message is to be sent. The **sendto** argument is the address of a descriptor pointing to this name.

The **sendto** argument is used in conjunction with the **sndtyp** argument. When **sndtyp** specifies BRK$C_DEVICE or BRK$C_USERNAME, the **sendto** argument is required.

If you do not specify **sndtyp** or if **sndtyp** does not specify BRK$C_DEVICE or BRK$C_USERNAME, you should not specify **sendto**; if **sendto** is specified, $BRKTHRU ignores it.

sndtyp

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

Terminal type to which $BRKTHRU is to send the message. The **sndtyp** argument is a longword value specifying the terminal type.

Each terminal type has a symbolic name, which is defined by the $BRKDEF macro. The following list describes each terminal type:

Terminal Type	Description
BRK$C_ALLUSERS	When specified, $BRKTHRU sends the message to all users who are currently logged in to the system.
BRK$C_ALLTERMS	When specified, $BRKTHRU sends the message to all terminals at which users are logged in and to all other terminals that are connected to the system except those with the AUTOBAUD characteristic set.
BRK$C_DEVICE	When specified, $BRKTHRU sends the message to a single terminal; you must specify the name of the terminal by using the **sendto** argument.

Terminal Type	Description
BRK$C_USERNAME	When specified, $BRKTHRU sends the message to a user with a specified user name; you must specify the user name by using the **sendto** argument.

iosb

VMS usage: **io_status_block**
type: **quadword (unsigned)**
access: **write only**
mechanism: **by reference**

I/O status block that is to receive the final completion status. The **iosb** is the address of this quadword block.

When the **iosb** argument is specified, $BRKTHRU sets the quadword to zero when it queues the message request. Then, after the message is sent to the specified terminal(s), $BRKTHRU returns four informational items, one item per word, in the quadword I/O status block.

These informational items indicate the status of the messages sent only to terminals and mailboxes on the local VAX node; these items do not include the status of messages sent to terminals and mailboxes on other VAX nodes in a VAXcluster.

The following lists, in order, each word of the quadword block and the informational item it contains:

Word	Informational Item
1	A condition value describing the final completion status.
2	A decimal number indicating the number of terminals and mailboxes to which $BRKTHRU successfully sent the message.
3	A decimal number indicating the number of terminals to which $BRKTHRU failed to send the message because the write to the terminal(s) timed out.
4	A decimal number indicating the number of terminals to which $BRKTHRU failed to send the message because the terminal(s) was set to the NOBROADCAST characteristic (by using the DCL command SET TERMINAL/NOBROADCAST).

carcon

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

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

For a list of the carriage control specifiers that you may use in the **carcon** argument, refer to the *VMS I/O User's Reference Volume*.

If you do not specify the **carcon** argument, $BRKTHRU uses a default value of 32, which represents a space in the ASCII character set. The message format resulting from this default value is a line feed, the message text, and a carriage return.

The **carcon** argument has no effect on message formatting specified by the BRK$M_SCREEN flag in the **flags** argument. See the description of the **flags** argument.

flags

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

Flag bit mask specifying options for the $BRKTHRU operation. The **flags** argument is a longword value that is the logical OR of each desired flag option.

Each flag option has a symbolic name. The $BRKDEF macro defines the following symbolic names:

Symbolic Name	Description
BRK$V_ERASE_LINES	When specified with the BRK$M_SCREEN flag, BRK$V_ERASE_LINES causes a specified number of lines to be cleared from the screen before the message is displayed. When BRK$M_SCREEN is not also specified, BRK$V_ERASE_LINES is ignored. Unlike the other Boolean flags, BRK$V_ERASE_LINES specifies a 1-byte integer in the range 0 to 24. It occupies the first byte in the longword flag mask. In coding the call to $BRKTHRU, specify the desired integer value in the OR operation with any other desired flags.
BRK$M_SCREEN	When specified, $BRKTHRU sends screen-formatted messages as well as messages formatted through the use of the **carcon** argument. $BRKTHRU sends screen-formatted messages to terminals with the DEC_CRT characteristic, and it sends messages formatted by **carcon** to those without the DEC_CRT characteristic. You set the DEC_CRT characteristic for the terminal by using the DCL command SET TERMINAL/DEC_CRT. A screen-formatted message is displayed at the top of the terminal screen, and the cursor is repositioned at the point it was prior to the broadcast message. However, the BRK$V_ERASE_LINES and BRK$M_BOTTOM flags also affect the display.
BRK$M_BOTTOM	When BRK$M_BOTTOM is specified and BRK$M_SCREEN is also specified, $BRKTHRU writes the message to the bottom of the terminal screen instead of the top. BRK$M_BOTTOM is ignored if the BRK$M_SCREEN flag is not set.

Symbolic Name	Description
BRK$M_NOREFRESH	When BRK$M_NOREFRESH is specified, $BRKTHRU, after writing the message to the screen, does not redisplay the last line of a read operation that was interrupted by the broadcast message. This flag is useful only when the BRK$M_SCREEN flag is not specified, because BRK$M_NOREFRESH is the default for screen-formatted messages.
BRK$M_CLUSTER	Specifying BRK$M_CLUSTER enables $BRKTHRU to send the message to terminals or mailboxes on other VAX nodes in a VAXcluster. If BRK$M_CLUSTER is not specified, $BRKTHRU sends messages only to terminals or mailboxes on the local VAX node.

reqid

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

Class requestor identification, which identifies to $BRKTHRU the application (or image) that is calling $BRKTHRU. The **reqid** argument is this longword identification value.

The **reqid** argument is used by several VMS images that send messages to terminals and may be used by as many as 16 different user images as well.

When such an image calls $BRKTHRU, specifying **reqid**, $BRKTHRU notifies the terminal that this image wants to write to the terminal. This makes it possible for you to allow the image to write or prevent it from writing to the terminal.

To prevent a particular image from writing to your terminal, you use the image's name in the DCL command SET TERMINAL /NOBROADCAST=image-name. Note that image-name in this DCL command is the same as the value of the **reqid** argument that the image passed to $BRKTHRU.

For example, you can prevent the VMS Mail Utility (which is an image) from writing to the terminal by issuing the DCL command SET BROADCAST=NOMAIL.

The $BRKDEF macro defines class names that are used by several VMS components. These components specify their class names by using the **reqid** argument in calls to $BRKTHRU. The $BRKDEF macro also defines 16 class names (BRK$C_USER1 through BRK$C_USER16) for the use of user images that call $BRKTHRU. The class names and the components to which they correspond are as follows:

Class Name	Component
BRK$C_GENERAL	This class name is used by (1) the VMS image invoked by the DCL command REPLY and (2) the callers of the $BRDCST service. This is the default if the **reqid** argument is not specified.

Class Name	Component
BRK$C_PHONE	This class name is used by the VMS Phone Facility.
BRK$C_MAIL	This class name is used by the VMS Mail Utility.
BRK$C_DCL	This class name is used by the Digital Command Language (DCL) interpreter for the CTRL/T command, which displays the process status.
BRK$C_QUEUE	This class name is used by the VMS queue manager, which manages print and batch jobs.
BRK$C_SHUTDOWN	This class name is used by the VMS system shutdown image, which is invoked by the DCL command REPLY/ID=SHUTDOWN.
BRK$C_URGENT	This class name is used by the VMS image invoked by the DCL command REPLY/ID=URGENT.
BRK$C_USER1 through	These class names can be used by user-written images.

timout

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

Timeout value, which is the number of seconds that must elapse before an attempted write by $BRKTHRU to a terminal is considered to have failed. The **timout** argument is this longword value (in seconds).

Because $BRKTHRU calls the $QIO service to perform writes to the terminal, the timeout value specifies the number of seconds allotted to $QIO to perform a single write to the terminal.

If you do not specify the **timout** argument, $BRKTHRU uses a default value of 0 seconds, which specifies infinite time (no timeout occurs).

The value specified by **timout** may be 0 or any number greater than 4; the numbers 1, 2, 3, and 4 are illegal.

When you press CTRL/S or the NO SCROLL key, $BRKTHRU cannot send a message to the terminal. In such a case, the value of **timout** is usually exceeded and the attempted write to the terminal fails.

astadr

VMS usage: **ast_procedure**
type: **procedure entry mask**
access: **call without stack unwinding**
mechanism: **by reference**

AST service routine to be executed after $BRKTHRU has sent the message to the specified terminal(s). The **astadr** argument is the address of the entry mask of this routine.

If you specify **astadr**, the AST routine executes at the same access mode as the caller of $BRKTHRU.

astprm

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

AST parameter to be passed to the AST routine specified by the **astadr** argument. The **astprm** argument specifies this longword parameter.

DESCRIPTION

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

The calling process must have WORLD privilege to send a message to a specific user by specifying the BRK$C_USERNAME symbolic code for the **sndtyp** argument.

The $BRKTHRU service permits the message text to be as long as 16,350 bytes; however, both the SYSGEN parameter MAXBUF and the caller's available process space may also affect the maximum length of the message text.

The $BRKTHRU service operates by assigning a channel (by using the $ASSIGN service) to the terminal and then writing to the terminal (by using the $QIO service). When calling $QIO, $BRKTHRU specifies the IO$_WRITEVBLK function code, together with the IO$M_BREAKTHRU, IO$M_CANCTRLO, and (optionally) IO$M_REFRESH function modifiers.

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

- If the terminal is performing a read operation 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).
- If the terminal is performing a write operation when $BRKTHRU sends the message, the message is displayed after the current write operation has completed.
- If the terminal has the NOBROADCAST characteristic set for all images, or if you have disabled the receiving of messages from the image that is issuing the $BRKTHRU call (see the description of the **reqid** argument), the message is not displayed.

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

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The message buffer, message buffer descriptor, device name string, or device name string descriptor cannot be read by the caller.
SS$_BADPARAM	The message length exceeds 16,350 bytes, the process's buffered I/O byte count limit (BYTLM) quota is insufficient, the message length exceeds the value specified by the SYSGEN parameter MAXBUF, the value of the TIMOUT parameter is nonzero and less than 4 seconds, the value of the REQID is outside the range 0 to 63, or the value of the SNDTYP is not one of the legal ones listed.
SS$_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	The system dynamic memory is insufficient for completing the request and the process has disabled resource wait mode with the Set Resource Wait Mode ($SETRWM) service.
SS$_NONLOCAL	The device is on a remote node.
SS$_NOOPER	The process does not have the necessary OPER privilege.
SS$_NOSUCHDEV	The specified terminal does not exist, or it cannot receive the message.

CONDITION VALUES RETURNED IN THE I/O STATUS BLOCK

Any condition values returned by the $ASSIGN, $FAO, $GETDVI, $GETJPI, or $QIO service.

$BRKTHRUW Breakthrough and Wait

The Breakthrough and Wait service sends a message to one or more terminals. The $BRKTHRUW service operates synchronously; that is, it returns to the caller after the message has been sent to the specified terminal(s).

For asynchronous operations, use the Breakthrough ($BRKTHRU) service; $BRKTHRU returns to the caller after queueing the message request, without waiting for the message to be delivered.

Aside from the preceding, $BRKTHRUW is identical to $BRKTHRU. For all other information about the $BRKTHRUW service, refer to the description of $BRKTHRU.

For additional information about system service completion, refer to the documentation of the Synchronize ($SYNCH) service and to the *Introduction to VMS System Services*.

The $BRKTHRU and $BRKTHRUW services supersede the Broadcast ($BRDCST) service. When writing new programs, you should use $BRKTHRU or $BRKTHRUW instead of $BRDCST. When updating old programs, you should change all uses of $BRDCST to $BRKTHRU or $BRKTHRUW. $BRDCST is now an obsolete system service and is no longer being enhanced.

FORMAT

SYS$BRKTHRUW *[efn],msgbuf[,sendto][,sndtyp][,iosb][,carcon][,flags][,reqid][,timout][,astadr][,astprm]*

$CANCEL Cancel I/O On Channel

The Cancel I/O On Channel service cancels all pending I/O requests on a specified channel. In general, this includes all I/O requests that are queued as well as the request currently in progress. To cancel I/O on a channel, the access mode of the calling process must be equal to or more privileged than the access mode of the process that made the original channel assignment.

FORMAT

SYS$CANCEL *chan*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENT

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

I/O channel on which I/O is to be canceled. The **chan** argument is a longword containing the channel number.

DESCRIPTION

The $CANCEL service requires system dynamic memory and uses the process's buffered I/O limit (BIOLM) quota.

When you cancel a request currently in progress, the driver is notified immediately. The actual cancellation may or may not occur immediately, depending on the logical state of the driver. When cancellation does occur, the following action for I/O in progress, similar to that for queued requests, takes place:

1. The specified event flag is set.
2. The first word of the I/O status block, if specified, is set to SS$_CANCEL if the I/O request is queued, or to SS$_ABORT if the I/O is in progress.
3. The AST, if specified, is queued.

Proper synchronization between this service and the actual canceling of I/O requests requires the issuing process to wait for I/O completion in the normal manner and then note that the I/O has been canceled.

If the I/O operation is a virtual I/O operation involving a disk or tape ACP, the I/O cannot be canceled. In the case of a magnetic tape, however, cancellation may occur if the device driver is hung.

Outstanding I/O requests are automatically canceled at image exit.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_EXQUOTA	The process has exceeded its buffered I/O limit (BIOLM) quota.
SS$_INSFMEM	The system dynamic memory is insufficient for canceling the I/O.
SS$_IVCHAN	You specified an invalid channel, 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.

$CANEXH Cancel Exit Handler

The Cancel Exit Handler service deletes an exit control block from the list of control blocks for the calling access mode. Exit control blocks are declared by the Declare Exit Handler ($DCLEXH) service and are queued according to access mode in a last-in first-out order.

FORMAT

SYS$CANEXH *[desblk]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENT

desblk
VMS usage: **exit_handler_block**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Control block describing the exit handler to be canceled. If you do not specify **desblk** or specify it as 0, all exit control blocks are canceled for the current access mode. The **desblk** argument is the address of this control block.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The first longword of the exit control block or the first longword of a previous exit control block in the list cannot be read by the caller, or the first longword of the preceding control block cannot be written by the caller.
SS$_IVSSRQ	The call to the service is invalid because it was made from kernel mode.
SS$_NOHANDLER	The specified exit handler does not exist.

$CANTIM Cancel Timer

The Cancel Timer Request service cancels all or a selected subset of the Set Timer requests previously issued by the current image executing in a process. Cancellation is based on the request identification specified in the Set Timer ($SETIMR) service. If you give the same request identification to more than one timer request, all requests with that request identification are canceled. The calling process can cancel only timer requests that are issued by a process whose access mode is equal to or less privileged than that of the calling process.

FORMAT

SYS$CANTIM *[reqidt],[acmode]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

reqidt

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

Request identification of the timer request(s) to be canceled. If you specify it as 0 (the default), all timer requests are canceled. The **reqidt** argument is a longword containing this identification.

acmode

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

Access mode of the request(s) to be canceled. 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$C_KERNEL	Kernel
PSL$C_EXEC	Executive
PSL$C_SUPER	Supervisor
PSL$C_USER	User

The most privileged access mode used is the access mode of the caller. Only those timer requests issued from an access mode equal to or less privileged than the resultant access mode are canceled.

DESCRIPTION

Canceled timer requests are restored to the process's quota for timer queue entries (TQELM quota).

Outstanding timer requests are automatically canceled at image exit.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.

$CANWAK Cancel Wakeup

The Cancel Wakeup service removes all scheduled wakeup requests for a process from the timer queue, including those made by the caller or by other processes. The Schedule Wakeup ($SCHDWK) service makes scheduled wakeup requests. Depending on the operation, use of $CANWAK may require the calling process to have a certain privilege. You need GROUP privilege to cancel wakeup requests issued for other processes in the same group, unless the process has the same UIC. You need WORLD privilege to cancel wakeup requests issued for any process in the system.

FORMAT

SYS$CANWAK *[pidadr] ,[prcnam]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

pidadr

VMS usage: **process_id**
type: **longword (unsigned)**
access: **modify**
mechanism: **by reference**

Process identification (PID) of the process for which wakeups are to be canceled. The **pidadr** argument is the address of a longword specifying the PID.

prcnam

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

Name of the process for which wakeups are to be canceled. The **prcnam** argument is the address of a character string descriptor pointing to the process name string.

VMS interprets the UIC group number of the calling process as part of the process name; the names of processes are unique to UIC groups. Because of this, you can use the **prcnam** argument only on behalf of processes in the same group as the calling process.

DESCRIPTION

Canceled wakeup requests are restored to the process's AST limit (ASTLM) quota.

If you specify neither the **pidadr** nor **prcnam** argument, scheduled wakeup requests for the calling process are canceled.

Pending wakeup requests issued by the current image are automatically canceled at image exit.

This service cancels only wakeup requests that have been scheduled; it does not cancel wakeup requests made with the Wake Process from Hibernation ($WAKE) service.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The process name string or string descriptor cannot be read by the caller, or the process identification cannot be written by the caller.
SS$_IVLOGNAM	The process name string has a length of 0 or has more than 15 characters.
SS$_NONEXPR	The specified process does not exist, or you specified an invalid process identification.
SS$_NOPRIV	The process does not have the privilege to cancel wakeups for the specified process.

$CHANGE_ACL Change Access Control List

The Change Access Control List service creates or modifies an object's access control list.

FORMAT

SYS$CHANGE_ACL *[chan] ,objtyp ,[objnam] ,itmlst ,[acmode] ,[nullarg] ,[contxt] ,[nullarg] ,[nullarg]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns 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 object whose ACL is modified when $CHANGE_ACL completes execution. The **chan** argument is a word containing the number of the channel. If you specify **objnam**, you must omit **chan** or specify it as zero.

objtyp

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

Type of object whose ACL is modified when $CHANGE_ACL completes execution. The **objtyp** argument is the address of a longword containing a value indicating whether the object is a file or a device. The symbols are defined in the system macro library ($ACLDEF). The values and their meanings are as follows.

Value	Meaning
ACL$C_DEVICE	Object is a device
ACL$C_FILE	Object is a Files–11 structure level 2 file
ACL$C_GROUP_GLOBAL_SECTION	Object is a group global section
ACL$C_JOBCTL_QUEUE	Object is a batch or print queue
ACL$C_LOGICAL_NAME_TABLE	Object is a logical name table
ACL$C_SYSTEM_GLOBAL_SECTION	Object is a system global section

objnam

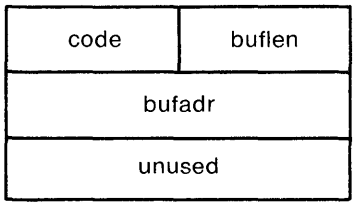
VMS usage: **char_string**
type: **character-coded text string**
access: **read only**
mechanism: **by descriptor—fixed-length string descriptor**

Name of the object whose ACL is modified when $CHANGE_ACL completes execution. The **objnam** argument is the address of a descriptor pointing to a character text string containing the name of the object. The maximum length of **objnam** depends on the object.

itmlst

VMS usage: **item_list_3**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Modifications to be made to the ACL when $CHANGE_ACL completes execution. The **itmlst** argument is the address of a variable length data structure defining the changes to be made. The data structure consists of three elements for each item code, as shown in the following diagram.



ZK-1701-84

buflen — Word containing the number of bytes in the buffer pointed to by **bufadr**

code — Word containing the item code

bufadr — Longword containing the address of a buffer for information being passed to or from $CHANGE_ACL

The third longword of the standard item list entry (the return length address) is not used by $CHANGE_ACL and should be zero.

End the item list with a longword containing the value zero.

The symbols for the item codes are defined in the system macro library ($ACLDEF). The values and their meanings are as follows:

Value	Meaning
ACL$C_ACLLENGTH	Returns the size, in bytes, of the object's ACL. The **bufadr** argument points to a longword that contains the size.
ACL$C_ADDACLENT	Adds an ACE to the beginning of the ACL when **contxt** is 0, to the end of the ACL when **contxt** is -1, or at a location pointed to by a prior ACL$C_FNDACETYP or ACL$C_FNDACLENT. The **bufadr** argument points to a variable-length data structure containing the ACE to be added. You can add more than one ACE with ACL$C_ADDACLENT; however, **buflen** must contain the total size of all ACEs contained in the buffer.
ACL$C_DELACLENT	Deletes the ACE pointed to by **bufadr**, or if **bufadr** is specified as zero, the ACE pointed to by a prior ACL$C_FNDACETYP or ACL$C_FNDACLENT.
ACL$C_DELETEACL	Deletes the entire ACL with the exception of protected ACEs.
ACL$C_FNDACETYP	Locates an ACE of the type pointed to by **bufadr**.
ACL$C_FNDACLENT	Locates the ACE pointed to by **bufadr**.
ACL$C_RLOCK_ACL	Obtains a read lock on an object in order to lock out all writers to the object's ACL. Regardless of the caller's mode, ACL locks are user-mode locks so that all access modes will interlock ACLs correctly.
ACL$C_WLOCK_ACL	Obtains an exclusive lock on an object in order to lock out all other readers and writers to the object's ACL. Regardless of the caller's mode, ACL locks are user-mode locks so that all access modes will interlock ACLs correctly.
ACL$C_MODACLENT	Replaces the ACE pointed to by a prior ACL$C_FNDACETYP or ACL$C_FNDACLENT with the ACE pointed to by **bufadr**.
ACL$C_READACE	Reads the ACE pointed to by ACL$C_FNDACETYP or ACL$C_FNDACLENT into the buffer pointed to by **bufadr**.
ACL$C_READACL	Reads the object's ACL. You should set the **contxt** argument initially to zero. Complete ACEs are read into the buffer pointed to by **bufadr**.
ACL$C_UNLOCK_ACL	Releases the lock obtained with ACL$C_RLOCK_ACL or ACL$C_WLOCK_ACL.

When you add an ACE with ACL$C_ADDACLENT or locate an ACE with ACL$C_FNDACETYP or ACL$C_FNDACLENT, $CHANGE_ACL searches the ACL for a match for the ACE in the ACE buffer. The $CHANGE_ACL service does not always make a match based on the entire ACE buffer; instead, the ACE type determines how $CHANGE_ACL makes a match. For example:

- A default protection ACE (ACE$C_DIRDEF) matches only on the type field (ACE$B_TYPE). An ACL can have only one default protection ACE because $CHANGE_ACL stops searching after it locates a match.
- An identifier ACE (ACE$C_KEYID) matches on the flags (ACE$W_FLAGS) and identifier (ACE$L_KEY) fields.
- An alarm ACE (ACE$C_ALARM) matches on the flags (ACE$W_FLAGS) and access mask (ACE$L_ACCESS) fields.
- All other ACE types match on the entire ACE buffer.

Because $CHANGE_ACL uses these matching rules, adding an ACE sometimes results in the replacement of another ACE. For example, if you add an identifier ACE with the same flags and identifier fields but a different access mask, the new ACE replaces the old ACE. When you add an ACE on the top of an ACL, $CHANGE_ACL deletes any matching ACE. If you add an ACE below a matching ACE in an ACL, the added ACE has no effect.

acmode

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

Access mode to use in checking file access protection. The **acmode** argument is the address of a longword containing the access mode. The **acmode** argument defaults to kernel mode; however, the system compares **acmode** against the caller's access mode and uses the least privileged mode.

The following access modes and their symbols are defined in the system macro library ($PSLDEF):

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

nullarg

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

Place-holding argument. This argument is reserved by DIGITAL.

contxt

VMS usage: **context**
type: **longword (unsigned)**
access: **modify**
mechanism: **by reference**

Context value that points to an ACE. The **contxt** argument is the address of a longword containing the context value.

DESCRIPTION

The Change Access Control List service creates or modifies an object's ACL. For information about the various types of ACLs and their associated formats, see the description of the $FORMAT_ACL service. For information about how to convert an ASCII string to an ACE, see the description of the $PARSE_ACL service.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The string or its descriptor cannot be read by the caller, or the buffer descriptor cannot be read by the caller, or the buffer cannot be written by the caller, or the buffer is too small to hold the ACL entry.
SS$_BADPARAM	You specified an invalid object type, attribute code, item size, or access mode.
SS$_INSFARG	The **objtyp** argument is not specified, or neither **chan** nor **objnam** is specified.
SS$_IVACL	The format of the access control list entry is invalid.

$CHECK_ACCESS Check Access

The Check Access service determines, on behalf of a third-party user, whether that user can access the object specified.

FORMAT

SYS$CHECK_ACCESS *objtyp ,objnam ,usrnam ,itmlst*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

objtyp

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

Type of object being accessed. The **objtyp** argument is the address of a longword containing a value specifying the type of object. The following symbols are defined in the system macro library ($ACLDEF):

Symbol	Meaning
ACL$C_DEVICE	Object is a device
ACL$C_FILE	Object is a Files–11 structure level 2 file
ACL$C_GROUP_GLOBAL_SECTION	Object is a group global section
ACL$C_SYSTEM_GLOBAL_SECTION	Object is a system global section
ACL$C_JOBCTL_QUEUE	Object is a batch or print queue
ACL$C_LOGICAL_NAME_TABLE	Object is a logical name table

objnam

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

Name of the object being accessed. The **objnam** argument is the address of a descriptor pointing to a character text string containing the name of the object. The maximum length of **objnam** depends on the object.

usrnam

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

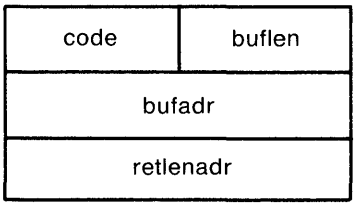
Name of the user attempting access. The **usrnam** argument is the address of a descriptor pointing to a character text string containing the user name of the user attempting to gain access to the specified object. The user name string may contain a maximum of 12 alphanumeric characters.

itmlst

VMS usage: **item_list_3**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Attributes describing how the object is to be accessed and information returned after $CHECK_ACCESS performs the protection check (for instance, security alarm information).

For each item code, you must include a set of four elements and end the list with a longword containing the value 0 (CHP$_END), as shown in the following diagram.



ZK-1703-84

buflen — Word containing the number of bytes in the buffer pointed to by **bufadr**.

code — Word containing the item code. The item codes are defined in the system macro library ($CHPDEF).

bufadr — Longword containing the address of a buffer used to pass information to or receive information from SYS$CHECK_ACCESS.

retlenadr — Longword containing the address of a word-long buffer in which SYS$CHECK_ACCESS writes the number of bytes written to the buffer pointed to by **bufadr**. If the buffer pointed to by **bufadr** is used to pass information to SYS$CHECK_ACCESS, **retlenadr** is ignored but must be included.

All items are optional. If you do not specify the access type item code (CHP$_ACCESS), read access is assumed.

The item codes used with $CHECK_ACCESS are described next. The first list of item codes defines the type of access desired. The second list of item codes allows you to determine which rights and privileges were used to access the object. The item codes are defined in the system macro library ($CHPDEF).

Input Items—Type of Access Desired

Item Identifier	Data Type	Description
CHP$_ACCESS	Longword	Bit mask representing the type of access desired ($ARMDEF)
CHP$_ACMODE	Byte	Accessor's processor access mode ($PSLDEF)
CHP$_FLAGS	Longword	Accessor's access to the object

CHP$_ACCESS

Only those bits set in CHP$_ACCESS are checked against the protection of the object to determine whether access is granted. (You can find the default definitions in the $ARMDEF macro.)

CHP$_ACMODE

The following access modes and their symbols are defined in the system macro library ($PSLDEF):

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

CHP$_FLAGS

The symbols in the next table are offsets to the bits within the longword. You can also obtain the values as masks with the appropriate bit set by using the prefix CHP$M rather than CHP$V. The following symbols are defined only in the system macro library ($CHPDEF):

Symbol	Access
CHP$V_READ	Accessor has read access.
CHP$V_WRITE	Accessor has write access.
CHP$V_USEREADALL	Accessor is eligible for READALL privilege.

Output Items—Information Returned

Item Identifier	Data Type	Description
CHP$_ALARMNAME	String	Character string containing the alarm name
CHP$_AUDITNAME	String	Character string containing the audit name
CHP$_MATCHEDACE	Block	The ACE in object's ACL that allowed or denied the accessor access to the object
CHP$_PRIVUSED	Longword	Mask of flags representing privileges used to gain the requested access

CHP$_ALARMNAME

If the object does not have security alarms enabled, SYS$CHECK_ACCESS returns **retlenadr** as 0.

CHP$_AUDITNAME

If the object does not have auditing enabled, SYS$CHECK_ACCESS returns **retlenadr** as 0.

CHP$_MATCHEDACE

This output item is a variable-length data structure containing the first identifier ACE in the object's ACL that allowed the accessor to access the object. The SYS$FORMAT_ACL system services describe the format of an identifier ACE.

CHP$_PRIVUSED

The following symbols are offsets to the bits within the longword:

Symbol	Meaning
CHP$_SYSPRV	SYSPRV was used to gain the requested access.
CHP$_GRPPRV	GRPPRV was used to gain the requested access.
CHP$_BYPASS	BYPASS was used to gain the requested access.
CHP$_READALL	READALL was used to gain the requested access.

You can also obtain the values as masks with the appropriate bit set by using the prefix CHP$M rather than CHP$V. The symbols are defined in the system macro library ($CHPDEF).

DESCRIPTION

The Check Access system service checks access to an object on behalf of a third-party process. One use of the $CHECK_ACCESS service might be for a file server that uses the service to check the protection attributes of a user (the third-party accessor) attempting access to a file (the object).

If the accessor can access the object, SYS$CHECK_ACCESS returns the SS$_NORMAL status code; otherwise, SYS$CHECK_ACCESS returns SS$_NOPRIV.

The arguments accepted by this service specify the name and type of object being accessed, the name of the user requesting access to the object, the type of access desired, and the type of information returned.

Alarm name strings are returned if an alarm record is to be written. A nonzero string length (as returned in the item descriptor) specifies the presence of an alarm request; if none is requested, a zero length is returned. Note that alarms may be requested whether the protection check succeeds or fails.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully; the desired access is granted.
SS$_NOPRIV	The desired access is not granted.
SS$_ACCVIO	The item list cannot be read by the caller, or one of the buffers specified in the item list cannot be written by the caller.

$CHKPRO Check Access Protection

The Check Access Protection service determines whether an accessor with the specified rights and privileges can access an object with the specified attributes.

FORMAT

SYS$CHKPRO *itmlst*

RETURNS

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

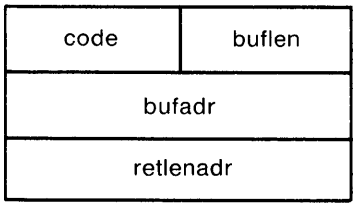
Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENT

itmlst
VMS usage: **item_list_3**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Protection attributes of the object and the rights and privileges of the accessor used when $CHKPRO determines if the accessor can access the object. The **itmlst** argument is the address of an item list of descriptors used to specify the protection attributes of the object and the rights and privileges of the accessor.

For each item code, you must include a set of four elements and end the list with a longword containing the value zero (CHP$_END), as shown in the following diagram.



ZK-1703-84

buflen — Word containing the number of bytes in the buffer pointed to by **bufadr**.

code — Word containing the item code. The item codes are defined in the system macro library ($ACLDEF).

bufadr — Longword containing the address of a buffer used to pass information to or receive information from SYS$CHKPRO.

retlenadr — Longword containing the address of a word-long buffer in which SYS$CHKPRO writes the number of bytes written to the buffer pointed to by **bufadr**. If the buffer pointed to by **bufadr** is used to pass information to SYS$CHKPRO, **retlenadr** is ignored but must be included.

All items are optional. Specifying any particular protection attribute causes that protection check to be made; any protection attribute not specified is not checked. Rights and privileges specified are used as needed. If a protection check requires any right or privilege not specified in the item list, the right or privilege of the caller's process is used.

The item codes used with $CHKPRO are described next. The first list of item codes defines the accessor's rights and privileges. The second list of item codes defines the object's protection attributes. The third list of item codes allows you to determine which rights and priviliges were used to access the object. The item codes are defined in the system macro library ($CHPDEF).

Input Items—Accessor's Rights and Privileges

Item Identifier	Data Type	Description
CHP$_ACCESS	Longword	Bit mask representing the type of access desired ($ARMDEF)
CHP$_ACMODE	Byte	Accessor's processor access mode
CHP$_ADDRIGHTS	Vector	Additional rights list segment to be appended to existing rights list
CHP$_FLAGS	Longword	Accessor's access to the object
CHP$_PRIV	Quadword	Accessor's privilege mask
CHP$_RIGHTS	Vector	Accessor's rights list

CHP$_ACCESS

Be aware that the $CHKPRO service does not interpret the bits in the access mask; instead, it compares them against the object's protection mask (CHP$_PROT). Any bits not specified by CHP$_ACCESS or CHP$_PROT are assumed to be clear, which grants access.

CHP$_ACMODE

The following access modes and their symbols are defined in the system macro library ($PSLDEF):

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

CHP$_ADDRIGHTS

Each entry of the rights list is a quadword data structure consisting of a longword containing the identifier value, followed by a longword containing a mask identifying the attributes of the holder. The SYS$CHKPRO service ignores the attributes.

A maximum of 11 rights descriptors is allowed. If you specify CHP$_ADDRIGHTS without specifying CHP$_RIGHTS, the accessor's rights list consists of the rights list specified by the CHP$_ADDRIGHTS item code(s) and the rights list of the current process.

If you specify CHP$_RIGHTS and CHP$_ADDRIGHTS, you should be aware of the following:

1. CHP$_RIGHTS must come first.
2. The accessor's UIC is the identifier of the first entry in the rights list specified by the CHP$_RIGHTS item code.
3. The accessor's rights list consists of the rights list specified by the CHP$_RIGHTS item code and the CHP$_ADDRIGHTS item code(s).

CHP$_FLAGS

The symbols in the next table are offsets to the bits within the longword. You can also obtain the values as masks with the appropriate bit set by using the prefix CHP$M rather than CHP$V. The following symbols are defined only in the system macro library ($CHPDEF).

Symbol	Access
CHP$V_READ	Accessor is making a read access
CHP$V_WRITE	Accessor is making a write access
CHP$V_USEREADALL	Accessor is eligible for READALL privilege

Because the access mask (CHP$_ACCESS) is not interpreted by $CHKPRO, CHP$FLAGS is used to determine whether the accessor is making a read or write access to the object, or both.

CHP$_PRIV

To form the symbolic names for the bits in the privilege mask, you must preface the name of the privileges with PRV$V_. For example, the bit associated with the BYPASS privilege is PRV$V_BYPASS. The privilege symbols are defined in the system macro library ($PRVDEF).

CHP$_RIGHTS

The accessor's UIC is the identifier of the first entry in the rights list. The accessor's rights list consists of the rights list specified by CHP$_RIGHTS and optionally the rights list specified by the CHP$_ADDRIGHTS item code(s).

Input Items—Object's Protection Attributes

Item Identifier	Data Type	Description
CHP$_ACL	Vector	Object's access control list
CHP$_MODE	Byte	Object's owner access mode
CHP$_MODES	Quadword	Object's access mode protection
CHP$_OWNER	Longword	Object's owner identifier (UIC or general identifier)
CHP$_PROT	Vector	Object's "SOGW" protection mask

CHP$_ACL

The buffer address, **bufadr**, specifies a buffer containing one or more ACEs. The number that specifies the length of the CHP$_ACL buffer, **buflen**, must be equal to the sum of all ACE lengths. The format of the ACE structure depends on the value of the second byte in the structure, which specifies the ACE type. The SYS$FORMAT_ACL system service description describes each ACE type and its format.

You may specify the CHP$_ACL item multiple times to point to multiple segments of an access control list. You may specify a maximum of 20 segments. The segments are processed in the order specified.

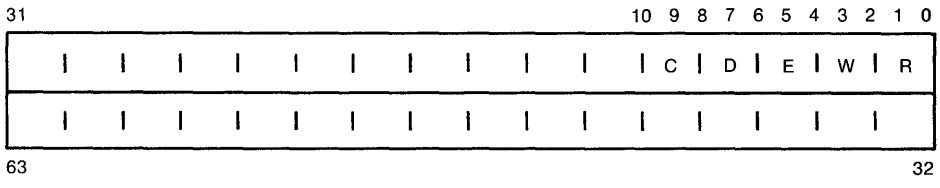
CHP$_MODE

The following access modes of the object's owner and their symbols are defined in the system macro library ($PSLDEF):

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

CHP$_MODES

You specify a 2-bit access mode as shown in CHP$_MODE for each possible access type. The following figure illustrates the format of an access mode vector.



ZK-1943-84

Each pair of bits in the access mode vector represents the access mode for the particular type of access. For example, bits <6:7> represent the access mode value used to check for delete access.

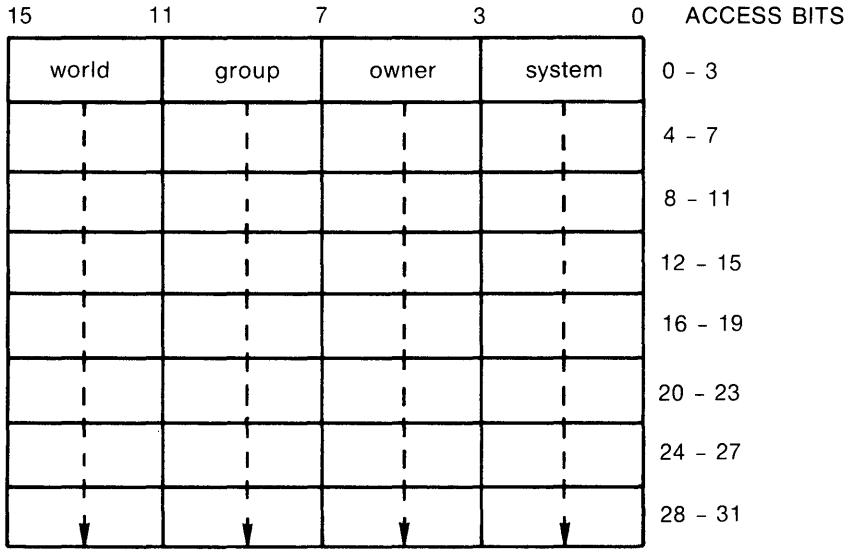
CHP$_OWNER

Specify a longword identifier indicating the owner of the object. This may be either a UIC format identifier or a general identifier.

Note: CHP$_OWNER is used in conjunction with the CHP$_PROT item code.

CHP$_PROT

The following diagram depicts the format for describing the object's protection.



The first word contains the first four protection bits for each field, the second word the next four protection bits, and so on. If a bit is clear, access is granted. By convention, the first five protection bits are (from right to left in each field of the first word) read, write, execute, delete, and (in the low-order bit in each field of the second word) control access. You may specify the CHP$_PROT item in increments of words; if a short buffer is given, zeroes are assumed for the remainder.

The $CHKPRO service compares the low-order four bits of CHP$_ACCESS against one of the four bit fields in the low-order word of CHP$_PROT, the next four bits of CHP$_ACCESS against one of the four bit fields in the next word of CHP$_PROT, and so on. The $CHKPRO service chooses a field of CHP$_PROT based on the privileges specified for the accessor (CHP$_PRIV), the UICs of the accessor (CHP$_RIGHTS or CHP$_ADDRIGHTS, or both), and the object's owner (CHP$_OWNER).

You must also specify the identifier of the object's owner with CHP$_OWNER when you use CHP$_PROT.

Output Items—Information Returned

Item Identifier	Data Type	Description
CHP$_ALARMNAME	String	Character string containing the alarm record
CHP$_MATCHEDACE	Block	Contains the ACE in object's ACL that allowed the accessor to access the object
CHP$_PRIVUSED	Longword	Mask of flags representing privileges used to gain the requested access

CHP$_ALARMNAME

If the object does not have security alarms enabled, SYS$CHKPRO returns **retlenadr** as zero.

CHP$_MATCHEDACE

This output item is a variable-length data structure containing the first identifier ACE in the object's ACL that allowed the accessor to access the object. The SYS$FORMAT_ACL system services describes the format of an identifier ACE.

CHP$_PRIVUSED

The following symbols are used as offsets to the bits within the longword:

Symbol	Meaning
CHP$_SYSPRV	Uses SYSPRV to gain the requested access
CHP$_GRPPRV	Uses GRPPRV to gain the requested access
CHP$_BYPASS	Uses BYPASS to gain the requested access
CHP$_READALL	Uses READALL to gain the requested access

You can also obtain the values as masks with the appropriate bit set by using the prefix CHP$M rather than CHP$V. The symbols are defined in the system macro library ($CHPDEF).

DESCRIPTION

The Check Access Protection service invokes the system's access protection check, allowing layered products and other subsystems to build protected structures within themselves whose protection is consistent with the protection facilities provided by the base system. It also allows a privileged subsystem to perform protection checks on behalf of some requester.

If the accessor can access the object, SYS$CHKPRO returns the SS$_NORMAL status code; otherwise, SYS$CHKPRO returns SS$_NOPRIV.

The arguments accepted by this service specify the protection of the object being accessed, the rights and privileges of the accessor, and the type of access desired. Because of the diverse and extensible nature of protections available in VMS, the arguments are presented in an item list.

When a protection check is to be invoked on behalf of another process, the privilege mask (CHP$_PRIV) is almost always mandatory, because it is used for all types of protection checks.

Alarm name strings are returned if an alarm record is to be written. A nonzero string length (as returned in the item descriptor) specifies the presence of an alarm request; if none is requested, a zero length is returned. Note that you may request alarms whether the protection check succeeds or fails.

For a flowchart detailing the operation of $CHKPRO, see the chapter on security services in the *Introduction to VMS System Services*.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully; the desired access is granted.
SS$_NOPRIV	The desired access is not granted.
SS$_ACCVIO	The item list cannot be read by the caller, or one of the buffers specified in the item list cannot be written by the caller.
SS$_BADPARAM	The argument is invalid.
SS$_IVACL	You supplied an invalid ACL segment with the CHP$_ACL item.

$CLREF Clear Event Flag

The Clear Event Flag service clears (sets to 0) an event flag in a local or common event flag cluster.

FORMAT

SYS$CLREF *efn*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENT

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

Number of the event flag to be cleared. The **efn** argument is a longword containing this number; however, $CLREF uses only the low-order byte.

CONDITION VALUES RETURNED

SS$_WASCLR	The service completed successfully. The specified event flag was previously 0.
SS$_WASSET	The service completed successfully. The specified event flag was previously 1.
SS$_ILLEFC	You specified an illegal event flag number.
SS$_UNASEFC	The process is not associated with the cluster containing the specified event flag.

$CMEXEC Change to Executive Mode

The Change to Executive Mode service changes the access mode of the calling process to executive mode. This service allows a process to change its access mode to executive, execute a specified routine, and then return to the access mode in effect before the call was issued.

FORMAT

SYS$CMEXEC *routin ,[arglst]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

routin

VMS usage: **procedure**
type: **procedure entry mask**
access: **call without stack unwinding**
mechanism: **by reference**

Routine to be executed while the process is in executive mode. The **routin** argument is the address of the entry point to this routine.

arglst

VMS usage: **arg_list**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Argument list to be passed to the routine specified by the **routin** argument. The **arglst** argument is the address of this argument list.

DESCRIPTION

To call this service, the process must either have CMEXEC or CMKRNL privilege or be currently executing in executive or kernel mode.

The $CMEXEC service uses standard procedure calling conventions to pass control to the specified routine. If no argument list is specified, the argument pointer (AP) contains a 0. However, to conform to the VAX Procedure Calling Standard, you must not omit the **arglist** argument.

When you use the $CMEXEC service, the system service dispatcher modifies both R0 and R1 before entry into the target routine. The specified routine must exit with a RET instruction and should place a status value in R0 before returning.

All of the Change Mode system services are intended to allow for the execution of a routine at an access mode more (not less) privileged than the access mode from which the call is made. If $CMEXEC is called while a process is executing in kernel mode, the routine specified by the **routin** argument executes in kernel mode, not executive mode.

CONDITION VALUES RETURNED

SS$_NOPRIV	The process does not have the privilege to change mode to executive.
All other values	The routine executed returns all other values.

$CMKRNL Change to Kernel Mode

The Change to Kernel Mode service changes the access mode of the calling process to kernel mode. This service allows a process to change its access mode to kernel, execute a specified routine, and then return to the access mode in effect before the call was issued.

FORMAT

SYS$CMKRNL *routin ,[arglst]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

routin

VMS usage: **procedure**
type: **procedure entry mask**
access: **call without stack unwinding**
mechanism: **by reference**

Routine to be executed while the process is in kernel mode. The **routin** argument is the address of the entry point to this routine.

arglst

VMS usage: **arg_list**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Argument list to be passed to the routine specified by the **routin** argument. The **arglst** argument is the address of this argument list.

DESCRIPTION

To call the $CMKRNL service, a process must either have CMKRNL privilege or be currently executing in executive or kernel mode.

The $CMKRNL service uses standard procedure calling conventions to pass control to the specified routine. If no argument list is specified, the argument pointer (AP) contains a 0. However, to conform to the VAX Procedure Calling Standard, you must not omit the **arglist** argument.

When you use the $CMKRNL service, the system service dispatcher modifies both R0 and R1 before entry into the target routine. The specified routine must exit with a RET instruction and should place a status value in R0 before returning.

The system loads R4 with the address of the Process Control Block (PCB).

CONDITION VALUES RETURNED

SS$_NOPRIV	The process does not have the privilege to change mode to kernel.
All other values	The routine executed returns all other values.

$CRELNM Create Logical Name

The Create Logical Name service creates a logical name and specifies its equivalence name(s).

FORMAT

SYS$CRELNM *[attr],tabnam,lognam,[acmode],[itmlst]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

attr

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

Attributes to be associated with the logical name. The **attr** argument is the address of a longword bit mask specifying these attributes.

Each bit in the longword corresponds to an attribute and has a symbolic name. These symbolic names are defined by the $LNMDEF macro. To specify an attribute, specify its symbolic name or set its corresponding bit. The longword bit mask is the logical OR of all desired attributes. All undefined bits in the longword must be 0.

If you do not specify this argument or specify it as 0 (no bits set), no attributes are associated with the logical name.

The attributes are as follows:

Attribute	Description
LNM$M_CONFINE	If set, the logical name is not copied from the process to its spawned subprocesses. You create a subprocess with the DCL command SPAWN or the LIB$SPAWN Run-Time Library routine. If the logical name is placed into a process-private table that has the CONFINE attribute, the CONFINE attribute is automatically associated with the logical name. This applies only to process-private logical names.
LNM$M_NO_ALIAS	If set, the logical name cannot be duplicated in this table at an outer access mode. If another logical name with the same name already exists in the table at an outer access mode, it is deleted.

tabnam

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

Name of the table in which to create the logical name. The **tabnam** argument is the address of a descriptor that points to the name of this table. This argument is required.

If **tabnam** is not the name of a logical name table, it is assumed to be a logical name and is translated iteratively until either the name of a logical name table is found or the number of translations allowed by the system has been performed. If **tabnam** translates to a list of logical name tables, the logical name is entered into the first table in the list.

You need the SYSNAM or SYSPRV privilege to specify the system table, and the GRPNAM or SYSPRV privilege to specify the group table.

You need the SYSPRV privilege to specify the system directory table LNM$SYSTEM_DIRECTORY.

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** argument is the address of a descriptor that points to the logical name string. Logical name strings of logical names created within either the system or process directory table must consist of alphanumeric characters, dollar signs, and underscores; the maximum length is 31 characters. The maximum length of logical name strings created within other tables is 255 characters with no restrictions on the types of characters that can be used. This argument is required.

acmode

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

Access mode to be associated with the logical name. The **acmode** argument is the address of a byte that specifies the access mode.

The access mode associated with the logical name is determined by "maximizing" the access mode of the caller with the access mode specified by the **acmode** argument, which means that the less privileged of the two is used. Symbols for the four access modes are defined by the $PSLDEF macro.

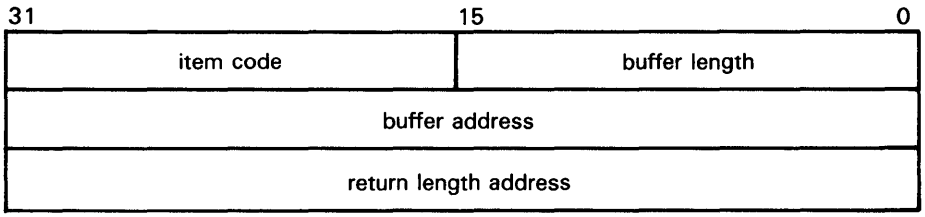
You cannot specify an access mode more privileged than that of the containing table. However, if the caller has SYSNAM privilege, then the specified access mode is associated with the logical name regardless of the access mode of the caller.

If you omit this argument or specify it as 0, the access mode of the caller is associated with the logical name.

itmlst

VMS usage: **item_list_3**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Item list describing the equivalence name(s) to be defined for the logical name and information to be returned to the caller. The **itmlst** argument is the address of a list of item descriptors, each of which specifies information about an equivalence name. The list of item descriptors is terminated by a longword of 0. This argument is required. The following diagram depicts a single item descriptor.



ZK-1705-84

$CRELNM Item Descriptor Fields

buffer length

A word specifying the number of bytes in the buffer pointed to by the **buffer address** field.

item code

A word that contains a symbolic code describing the nature of the information in the buffer or to be returned to the buffer pointed to by the **buffer address** field. The item codes are described under "$CRELNM Item Codes."

buffer address

A longword containing the address of the buffer that receives or passes information.

return length address

A longword containing the address of a word that receives the actual length in bytes of the information returned by $CRELNM in the buffer pointed to by the **buffer address** field. The **return length address** field is used only when the item code specified is LNM$_TABLE. Although this field is ignored for all other item codes, it must nevertheless be present as a placeholder in each item descriptor:

$CRELNM Item Codes

LNM$_ATTRIBUTES

When you specify LNM$_ATTRIBUTES, the **buffer address** field of the item descriptor points to a longword bit mask that specifies the current translation attributes for the logical name. The current translation attributes are applied to all subsequently specified equivalence strings until another LNM$_ATTRIBUTES item descriptor is encountered in the item list. The symbolic names for these attributes are defined by the $LNMDEF macro. The symbolic name and description of each attribute are as follows:

Attribute	Description
LNM$M_CONCEALED	If set, RMS interprets the equivalence name as a device name or logical name with the LNM$M_CONCEALED attribute.
LNM$M_TERMINAL	If set, further iterative logical name translation on the equivalence name is not to be performed.

LNM$_CHAIN

When you specify LNM$_CHAIN, the **buffer address** field of the item descriptor points to another item list that $CRELNM is to process immediately after it has processed the current item list.

If you specify the LNM$_CHAIN item code, it must be the last item code in the current item list.

LNM$_STRING

When you specify LNM$_STRING, the **buffer address** field of the item descriptor points to a buffer containing a user-specified equivalence name for the logical name. The maximum length of the equivalence string is 255 characters.

When $CRELNM encounters an item descriptor with the item code LNM$_STRING, it creates an equivalence name entry for the logical name using the most recently specified values for LNM$_ATTRIBUTES. The equivalence name entry includes the following information:

- The name specified by LNM$_STRING.
- The next available index value. Each equivalence is assigned a unique value from 0 to 127.
- The attributes specified by the most recently encountered item descriptor with item code LNM$_ATTRIBUTES (if these are present in the item list).

Therefore, you should construct the item list so that the LNM$_ATTRIBUTES item codes immediately precede the LNM$_STRING item code or codes to which they apply.

LNM$_TABLE

When you specify LNM$_TABLE, the **buffer address** field of the item descriptor points to a buffer in which $CRELNM writes the name of the logical name table in which it entered the logical name. The **return length address** field points to a word that contains a buffer that specifies the length in bytes of the information returned by $CRELNM. The maximum length of the name of a logical name table is 31 characters.

This item code may appear anywhere in the item list.

DESCRIPTION

The calling process must have the following:

- Write access to shareable tables to create logical names in those tables
- SYSNAM privilege to create executive or kernel mode logical names
- GRPNAM or SYSPRV privilege to enter a logical name into the group logical name table
- SYSNAM or SYSPRV privilege to enter a logical name into the system logical name table

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully; the logical name has been created.
SS$_SUPERSEDE	The service completed successfully; the logical name has been created and a previously existing logical name with the same name has been deleted.
SS$_BUFFEROVF	The service completed successfully; the **buffer length** field in an item descriptor specified an insufficient value, so the buffer was not large enough to hold the requested data.
SS$_ACCVIO	The service cannot access the location(s) specified by one or more arguments.

SS$_BADPARAM	One or more arguments have an invalid value, or a logical name table name or logical name was not specified.
SS$_DUPLNAM	An attempt was made to create a logical name with the same name as an already existing logical name, and the existing logical name was created at a more privileged access mode and with the LNM$M_NO_ALIAS attribute.
SS$_EXLNMQUOTA	The quota associated with the specified logical name table for the creation of the logical name is insufficient.
SS$_INSFMEM	The dynamic memory is insufficient for the creation of the logical name.
SS$_IVLOGNAM	The **tabnam** argument, **lognam** argument, or the equivalence string specifies a string whose length is not in the required range of 1 through 255 characters. The **lognam** argument specifies a string whose length is not in the required range of 1 to 31 characters for directory table entries.
SS$_IVLOGTAB	The **tabnam** argument does not specify a logical name table.
SS$_NOLOGTAB	Either the specified logical name table does not exist or the logical name translation of the table name exceeded the allowable depth of 10 translations.
SS$_NOPRIV	The caller lacks the necessary privilege to create the logical name.

$CRELNT Create Logical Name Table

The Create Logical Name Table service creates a process-private or shareable logical name table.

FORMAT

SYS$CRELNT *[attr] ,[resnam] ,[reslen] ,[quota] ,[promsk] ,[tabnam] ,partab ,[acmode]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

attr

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

Attributes to affect the creation of the logical name table and to be associated with the newly created logical name table. The **attr** argument is the address of a longword bit mask specifying these attributes.

Each bit in the longword corresponds to an attribute and has a symbolic name. These symbolic names are defined by the $LNMDEF macro. To specify an attribute, specify its symbolic name or set its corresponding bit. The longword bit mask is the logical OR of all desired attributes. All unused bits in the longword must be 0.

If you do not specify this argument or specify it as 0 (no bits set), no attributes are associated with the logical name table or affect the creation of the new table.

The following list describes each attribute:

Attribute	Description
LNM$M_CONFINE	If set, the logical name table is not copied from the process to its spawned subprocesses. You create a subprocess with the DCL command SPAWN or the Run-Time Library LIB$SPAWN routine. You may specify this attribute only for process-private logical name tables; it is ignored for shareable tables.

Attribute	Description
	The state of this bit is also propagated from the parent table to the newly created table and can be overridden only if the parent table does not have the bit set. Thus, if the parent table has the LNM$M_CONFINE attribute, the newly created table will also have it, no matter what is specified in the **attr** argument. On the other hand, if the parent table does not have the LNM$M_CONFINE attribute, the newly created table can be given this attribute through the **attr** argument. The process-private directory table LNM$PROCESS_DIRECTORY does not have the LNM$M_CONFINE attribute.
LNM$M_CREATE_IF	If set, a new logical name table is created only if the specified table name is not already entered at the specified access mode in the appropriate directory table. If the table name exists, a new table is not created and no modification is made to the existing table name. This holds true even if the existing name has differing attributes or quota values, or even if it is not the name of a logical name table. If LNM$M_CREATE_IF is not set, the new logical name table will supersede any existing table name with the same access mode within the appropriate directory table. Setting this attribute is useful when two or more users want to create and use the same table but do not want to synchronize its creation.
LNM$M_NO_ALIAS	If set, the name of the logical name table cannot be duplicated at an outer access mode within the appropriate directory table. If this name already exists at an outer access mode, it is deleted.

resnam

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

Name of the newly created logical name table, returned by $CRELNT. The **resnam** argument is the address of a descriptor pointing to this name. The name is a character string whose maximum length is 31 characters.

reslen

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

Length in bytes of the name of the newly created logical name table, returned by $CRELNT. The **reslen** argument is the address of a word to receive this length.

quota

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

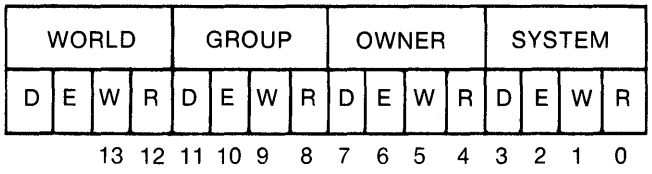
Maximum number of bytes of memory to be allocated for logical names contained in this logical name table. The **quota** argument is the address of a longword specifying this value.

If you specify no quota value, the logical name table has an infinite quota. Note that a shareable table created with infinite quota permits users with write access to that table to consume system dynamic memory without limit.

promsk

VMS usage: **file_protection**
type: **word (unsigned)**
access: **read only**
mechanism: **by reference**

Protection mask to be associated with the newly created shareable logical name table. The **promsk** argument is the address of a word that contains a value that represents four 4-bit fields, where each field describes the type of access allowed for system, owner, group, and world users. The following diagram depicts these protection bits.



ZK-1706-84

Each field consists of four bits specifying protection for the logical name table. The remaining bits in the protection mask are as follows:

- Read privileges allow access to names in the logical name table.
- Write privileges allow creation and deletion of names within the logical name table.
- Delete privileges allow deletion of the logical name table.

Note: The "E" protection bit is reserved by DIGITAL.

If a bit is clear, access is granted. If you omit the mask, complete access is granted to system and owner, and no access is granted to world and group.

tabnam

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

The name of the new logical name table. The **tabnam** argument is the address of a character string descriptor pointing to this name string. Table names are contained in either the process or system directory table

(LNM$PROCESS_DIRECTORY or LNM$SYSTEM_DIRECTORY). Therefore, table names must consist of alphanumeric characters, dollar signs ($), and underscores (_); the maximum length is 31 characters.

If you do not specify this argument, a default name in the format LNM$xxxx is used, where xxxx is a unique hexadecimal number.

You need SYSPRV privilege to specify the name of a shareable logical name table.

partab

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

Name string for the parent table name. The **partab** argument is the address of a character string descriptor pointing to this name string. If the parent table is shareable, then the newly created table is shareable and is entered into the system directory LNM$SYSTEM_DIRECTORY. If the parent table is process-private, then the newly created table is process-private and is entered in the process directory LNM$PROCESS_DIRECTORY. You need SYSPRV privilege or write access to the system directory to create a named shareable table. This argument is required.

acmode

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

Access mode to be associated with the newly created logical name table. The **acmode** argument is the address of a byte containing this access mode. The $PSLDEF macro defines symbolic names for the four access modes.

If you do not specify the **acmode** argument or specify it as 0, the access mode of the caller is associated with the newly created logical name table.

The access mode associated with the logical name table is determined by "maximizing" the access mode of the caller with the access mode specified by the **acmode**. The less privileged of the two access modes is used.

However, if the caller has SYSNAM privilege, then the specified access mode is associated with the logical name table, regardless of the access mode of the caller.

Access modes associated with logical name tables govern logical name table processing and provide a protection mechanism that prevents the deletion of inner access mode logical name tables by nonprivileged users. You cannot specify an access mode more privileged than that of the parent table.

A logical name table with supervisor-mode access may contain supervisor-mode and user-mode logical names and may be a parent to supervisor-mode and user-mode logical name tables, but may not contain executive- or kernel-mode logical names or be a parent to executive- or kernel-mode logical name tables.

You need SYSNAM privilege to specify executive- or kernel-mode access for a logical name table.

DESCRIPTION

Depending on the operation, use of $CRELNT may require the calling process to have certain privileges:

- You need the SYSPRV privilege to create a shareable table.
- You need the SYSNAM privilege to create a table at an access mode more privileged than that of the calling process.

The $CRELNT service uses the following system resources:

- System paged dynamic memory to create a shareable logical name table
- Process dynamic memory to create a process-private logical name table

The parent table governs whether the new table is process-private or shareable. If the parent table is process-private, so is the new table; if the parent table is shareable, so is the new table.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully; the logical name table already exists.
SS$_LNMCREATED	The service completed successfully; the logical name table was created.
SS$_SUPERSEDE	The service completed successfully; the logical name table was created and its logical name superseded already existing logical name(s) in the directory table.
SS$_ACCVIO	The service cannot access the location(s) specified by one or more arguments.
SS$_BADPARAM	One or more arguments has an invalid value, or a parent logical name table was not specified.
SS$_DUPLNAM	You attempted to create a logical name table with the same name as an already existing name within the appropriate directory table, and the existing name was created at a more privileged access mode with the LNM$M_NO_ALIAS attribute.
SS$_EXLNMQUOTA	The parent table has insufficient quota for the creation of the new table.
SS$_INSFMEM	The dynamic memory is insufficient for the creation of the table.
SS$_IVLOGNAM	The **partab** argument specifies a string whose length is not within the required range of 1 to 31 characters.
SS$_IVLOGTAB	The **tabnam** argument is not alphanumeric or specifies a string whose length is not within the required range of 1 to 31 characters.
SS$_NOLOGTAB	The parent logical name table does not exist.
SS$_NOPRIV	The caller lacks the necessary privilege to create the table.

SS$_PARENT_DEL	The creation of the new table would have resulted in the deletion of the parent table.
SS$_RESULTOVF	The table name buffer is not large enough to contain the name of the new table.

$CREATE_RDB Create Rights Database

The Create Rights Database service initializes a rights database.

FORMAT

SYS$CREATE_RDB *[sysid]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENT

sysid
VMS usage: **system_access_id**
type: **quadword (unsigned)**
access: **read only**
mechanism: **by reference**

System identification value associated with the rights database when $CREATE_RDB completes execution. The **sysid** argument is the address of quadword containing the system identification value. If you omit **sysid**, the current system time in 64-bit format is used.

DESCRIPTION

The Create Rights Database service initializes a rights database. The database name is the file equated to the logical name RIGHTSLIST, which must be defined as a system logical name at executive mode. If the logical name does not exist, the database is named SYS$SYSTEM:RIGHTSLIST.DAT.

If the database already exists, $CREATE_RDB fails with the error RMS$_FEX.

You need write access to the rights database to use this service. If the database is in SYS$SYSTEM (which is the default), you need SYSPRV privilege to grant write access to the database.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The **sysid** argument cannot be read by the caller.
SS$_INSFMEM	The process dynamic memory is insufficient for opening the rights database.

RMS$_FEX	A rights database already exists. To create a new one, you must explicitly delete or rename the old one.
RMS$_PRV	The user does not have write access to SYS$SYSTEM.

Because the rights database is an indexed file accessed with VMS RMS, this service may also return RMS status codes associated with operations on indexed files. For descriptions of these status codes, refer to the *VMS Record Management Services Manual.*

$CREMBX Create Mailbox and Assign Channel

The Create Mailbox and Assign Channel service creates a virtual mailbox device named MBA*n* and assigns an I/O channel to it. The system provides the unit number *n* when it creates the mailbox. If a mailbox with the specified name already exists, the $CREMBX service assigns a channel to the existing mailbox.

FORMAT

SYS$CREMBX *[prmflg] ,chan ,[maxmsg] ,[bufquo] ,[promsk] ,[acmode] ,[lognam]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

prmflg

VMS usage: **boolean**
type: **byte (unsigned)**
access: **read only**
mechanism: **by value**

Indicator specifying whether the created mailbox is to be permanent or temporary. The **prmflg** argument is a byte value. The value 1 specifies a permanent mailbox; the value 0, which is the default, specifies a temporary mailbox. Any other values result in an error.

chan

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

Channel number assigned by $CREMBX to the mailbox. The **chan** argument is the address of a word into which $CREMBX writes the channel number.

maxmsg

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

Maximum size (in bytes) of a message that can be sent to the mailbox. The **maxmsg** argument is a longword value containing this size. If you do not specify **maxmsg** or you specify it as 0, VMS provides a default value.

bufquo

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

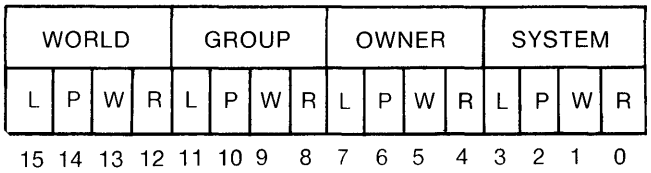
Number of bytes of system dynamic memory that can be used to buffer messages sent to the mailbox. The **bufquo** argument is a longword value containing this number. If you do not specify the **bufquo** argument or you specify it as 0, VMS provides a default value.

The maximum value you can specify with the **bufquo** argument is 65355. For a temporary mailbox, this value must be less than or equal to the process buffer quota.

promsk

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

Protection mask to be associated with the created mailbox. The **promsk** argument is a longword value that is the combined value of the bits set in the protection mask. Cleared bits grant access and set bits deny access to each of the four classes of user: world, group, owner, and system. The following diagram depicts these protection bits.



ZK-1707-84

If you do not specify the **promsk** argument or you specify it as 0, read, write, physical, and logical access are granted to all users.

The physical access bit is ignored for all categories of user. The logical access bit must be clear for all categories of user because logical access is required to read or write to a mailbox; thus, setting or clearing the read and write access bits is meaningless unless the logical access bit is also cleared.

Logical access also allows you to queue read or write attention ASTs.

acmode

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

Access mode to be associated with the channel to which the mailbox is assigned. 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$C_KERNEL	Kernel
PSL$C_EXEC	Executive
PSL$C_SUPER	Supervisor
PSL$C_USER	User

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

lognam

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

Logical name to be assigned to the mailbox. The **lognam** argument is the address of a character string descriptor pointing to the logical name string.

The equivalence name for the mailbox is MBA*n*. The equivalence name is marked with the terminal attribute. Processes can use the logical name to assign other I/O channels to the mailbox.

For permanent mailboxes, the $CREMBX service enters the specified logical name, if any, in the LNM$PERMANENT_MAILBOX logical name table and, for temporary mailboxes, into the LNM$TEMPORARY_MAILBOX logical name table.

DESCRIPTION

Depending on the operation, the calling process may need one of the following privileges to use $CREMBX:

- TMPMBX privilege to create a temporary mailbox
- PRMMBX privilege to create a permanent mailbox
- SYSNAM privilege to place a logical name for a mailbox in the system logical name table
- GRPNAM privilege to place a logical name for a mailbox in the group logical name table

The $CREMBX service uses system dynamic memory to allocate a device database for the mailbox and for an entry in the logical name table (if a logical name is specified).

When a temporary mailbox is created, the process's buffered I/O byte count (BYTLM) quota is reduced by the amount specified in the **bufquo** argument. The size of the mailbox unit control block and the logical name (if specified) are also subtracted from the quota. The quota is returned to the process when the mailbox is deleted.

After the process creates a mailbox, it and other processes can assign additional channels to it by calling the Assign I/O Channel ($ASSIGN) or Create Mailbox ($CREMBX) service. If the mailbox already exists, the $CREMBX service assigns a channel to that mailbox; in this way, cooperating processes need not consider which process must execute first to create the

mailbox. The system maintains a reference count of the number of channels assigned to a mailbox, and this count is decreased whenever a channel is deassigned with the Deassign I/O Channel ($DASSGN) service or when the image that assigned the channel exits.

A temporary mailbox is deleted when no more channels are assigned to it. A permanent mailbox must be explicitly marked for deletion with the Delete Mailbox ($DELMBX) service; its actual deletion occurs when no more channels are assigned to it.

A mailbox is treated as a shareable device; it cannot, however, be mounted or allocated.

Mailboxes are assigned sequentially increasing unit numbers (from 1 to a maximum of 9999) as they are created. When all unit numbers have been used, the system starts numbering again at unit 1.

A process can obtain the unit number of the created mailbox by calling the Get Device/Volume Information ($GETDVI) service.

Default values for the maximum message size and the buffer quota (an appropriate multiple of the message size) are determined for a specific system during system generation. The SYSGEN parameter DEFMBXMXMSG determines the maximum message size; the SYSGEN parameter DEFMBXBUFQUO determines the buffer quota. For termination mailboxes, the maximum message size must be at least as large as the termination message (currently 84 bytes).

When you specify a logical name for a temporary mailbox, the $CREMBX service enters the name into the LNM$TEMPORARY_MAILBOX logical name table.

Normally, LNM$TEMPORARY_MAILBOX specifies LNM$JOB, the job-wide logical name table; thus, only processes in the same job as the process that first creates the mailbox can use the logical name to access the temporary mailbox. If you want to use the temporary mailbox to enable communication between processes in different jobs, you must redefine LNM$TEMPORARY_MAILBOX in the process logical name directory table (LNM$PROCESS_DIRECTORY) to specify a logical name table that those processes can access.

For instance, if you want to use the mailbox as a communication device for processes in the same group, you must redefine LNM$TEMPORARY_MAILBOX to specify LNM$GROUP, the group logical name table. The following DCL command assigns temporary mailbox logical names to the group logical name table:

```
$ DEFINE/TABLE=LNM$PROCESS_DIRECTORY-
_$ LNM$TEMPORARY_MAILBOX  LNM$GROUP
```

When you specify a logical name for a permanent mailbox, the system enters the name in the logical name table specified by the logical name table name LNM$PERMANENT_MAILBOX, which normally specifies LNM$SYSTEM, the system logical name table. If you want the logical name that you specify for the mailbox to be entered in a logical name table other than the system logical name table, you must redefine LNM$PERMANENT_MAILBOX to specify the desired table. For more information about logical name tables, see the *Introduction to VMS System Services*.

If you redefine either LNM$TEMPORARY_MAILBOX or LNM$PERMANENT_MAILBOX, be sure that the name of the new table appears in the logical name table LNM$FILE_DEV. RMS and the I/O system services use LNM$FILE_DEV to translate I/O device names. If the logical name table specified by either LNM$TEMPORARY_MAILBOX or LNM$PERMANENT_MAILBOX does not appear in LNM$FILE_DEV, the system will be unable to translate the logical name of your mailbox and therefore will be unable to access your mailbox as an I/O device.

If you redirect a logical name table to point to a process-private table, then the following occurs:

- Other processes cannot access the mailbox by its name.
- If the creating process issues a second call to $CREMBX, a different mailbox is created and a channel is assigned to the new mailbox. (If the creating process issues a second call to $CREMBX using a shared logical name, a second channel is assigned to the existing mailbox.)
- The logical name is not deleted when the mailbox disappears.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The logical name string or string descriptor cannot be read by the caller, or the channel number cannot be written by the caller.
SS$_BADPARAM	The **bufquo** argument specified a value greater than approximately 65355, which is 65535 minus the size of a mailbox unit control block (UCB).
SS$_EXBYTLM	The process has insufficient buffer I/O byte count (BYTLM) quota to allocate the mailbox UCB or to satisfy buffer requirements.
SS$_INSFMEM	The system dynamic memory is insufficient for completing the service.
SS$_INTERLOCK	The bit map lock for allocating mailboxes from the specified shared memory is locked by another process.
SS$_IVLOGNAM	The logical name string has a length of 0 or has more than 255 characters.
SS$_IVSTSFLG	The bit set in the **prmflg** argument is undefined; this argument may have a value of 1 or 0.
SS$_NOIOCHAN	No I/O channel is available for assignment.
SS$_NOPRIV	The process does not have the privilege to create a temporary mailbox, a permanent mailbox, a mailbox in memory that is shared by multiple processors, or a logical name.
SS$_NOSHMBLOCK	No shared memory mailbox control block is available for use to create a new mailbox.

SS$_OPINCOMPL	A duplicate unit number was encountered while linking a shared memory mailbox UCB. If this condition value is returned, submit an SPR to DIGITAL.
SS$_SHMNOTCNCT	The shared memory named in the **lognam** argument is not known to the system. This error can be caused by a spelling error in the string, an improperly assigned logical name, or the failure to identify the memory as shared at system generation time.
SS$_TOOMANYLNAM	The logical name translation of the string named in the **lognam** argument exceeded the allowed depth.

$CREPRC Create Process

The Create Process service creates a subprocess or detached process on behalf of the calling process.

FORMAT

SYS$CREPRC *[pidadr] ,[image] ,[input] ,[output] ,[error] ,[prvadr] ,[quota] ,[prcnam] ,[baspri] ,[uic] ,[mbxunt] ,[stsflg]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

pidadr

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

Process identification (PID) of the newly created process. The **pidadr** argument is the address of a longword into which $CREPRC writes the PID.

image

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

Name of the image to be activated in the newly created process. The **image** argument is the address of a character string descriptor pointing to the file specification of the image.

The image name can have a maximum of 63 characters. If the image name contains a logical name, the equivalence name must be in a logical name table that the created process can access.

input

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

Equivalence name to be associated with the logical name SYS$INPUT in the logical name table of the created process. The **input** argument is the address of a character string descriptor pointing to the equivalence name string.

output

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

Equivalence name to be associated with the logical name SYS$OUTPUT in the logical name table of the created process. The **output** argument is the address of a character string descriptor pointing to the equivalence name string.

error

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

Equivalence name to be associated with the logical name SYS$ERROR in the logical name table of the created process. The **error** argument is the address of a character string descriptor pointing to the equivalence name string.

Note that the **error** argument is ignored if the **image** argument specifies SYS$SYSTEM:LOGINOUT.EXE; in this case, SYS$ERROR points to SYS$OUTPUT.

prvadr

VMS usage: **mask_privileges**
type: **quadword (unsigned)**
access: **read only**
mechanism: **by reference**

Privileges to be given to the created process. The **prvadr** argument is the address of a quadword bit vector wherein each bit corresponds to a privilege; setting a bit gives the privilege.

Each bit has a symbolic name; the $PRVDEF macro defines these names. You form the bit vector by specifying the symbolic name of each desired privilege in a logical OR operation. Table SYS–1 gives the symbolic name and description of each privilege.

Table SYS–1 User Privileges

Privilege	Symbolic Name	Description
ALLSPOOL	PRV$V_ALLSPOOL	Allocate a spooled device
BUGCHK	PRV$V_BUGCHK	Make bugcheck error log entries
BYPASS	PRV$V_BYPASS	Bypass UIC-based protection
CMEXEC	PRV$V_CMEXEC	Change mode to executive
CMKRNL	PRV$V_CMKRNL	Change mode to kernel
DETACH	PRV$V_DETACH	Create detached processes
DIAGNOSE	PRV$V_DIAGNOSE	May diagnose devices
DOWNGRADE	PRV$V_DOWNGRADE	May downgrade classification
EXQUOTA	PRV$V_EXQUOTA	May exceed quotas

Table SYS–1 (Cont.) User Privileges

Privilege	Symbolic Name	Description
GROUP	PRV$V_GROUP	Group process control
GRPNAM	PRV$V_GRPNAM	Place name in group logical name table
GRPPRV	PRV$V_GRPPRV	Group access via system protection field
LOG_IO	PRV$V_LOG_IO	Perform logical I/O operations
MOUNT	PRV$V_MOUNT	Issue mount volume QIO
NETMBX	PRV$V_NETMBX	Create a network device
ACNT	PRV$V_NOACNT	Create processes for which no accounting is done
OPER	PRV$V_OPER	All operator privileges
PFNMAP	PRV$V_PFNMAP	Map to section by physical page frame number
PHY_IO	PRV$V_PHY_IO	Perform physical I/O operations
PRMCEB	PRV$V_PRMCEB	Create permanent common event flag clusters
PRMGBL	PRV$V_PRMGBL	Create permanent global sections
PRMMBX	PRV$V_PRMMBX	Create permanent mailboxes
PSWAPM	PRV$V_PSWAPM	Change process swap mode
READALL	PRV$V_READALL	Possess read access to everything
SECURITY	PRV$V_SECURITY	May perform security functions
ALTPRI	PRV$V_SETPRI	Set (alter) any process priority
SETPRV	PRV$V_SETPRV	Set any process privileges
SHARE	PRV$V_SHARE	May assign a channel to a non-shared device
SYSGBL	PRV$V_SYSGBL	Create system global sections
SYSLCK	PRV$V_SYSLCK	Queue system-wide locks
SYSNAM	PRV$V_SYSNAM	Place name in system logical name table
SYSPRV	PRV$V_SYSPRV	Access files and other resources as if you have a system UIC
TMPMBX	PRV$V_TMPMBX	Create temporary mailboxes
UPGRADE	PRV$V_UPGRADE	May upgrade classification
VOLPRO	PRV$V_VOLPRO	Override volume protection
WORLD	PRV$V_WORLD	World process control

Note that the names of the privilege bits PRV$V_NOACNT and PRV$V_SETPRI correspond to the names of the DCL privileges ACNT and ALTPRI, yet have different names.

You need the user privilege SETPRV to grant a process any privileges other than your own. If the caller does not have this privilege, the mask is minimized with the current privileges of the creating process; any privileges the creator does not have are not granted, but no error status code is returned.

quota

VMS usage: **item_quota_list**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Process quotas to be established for the created process. These quotas limit the created process's use of system resources. The **quota** argument is the address of a list of quota descriptors, where each quota descriptor consists of a 1-byte quota name followed by a longword that specifies the desired value for that quota. The list of quota descriptors is terminated by the symbolic name PQL$_LISTEND.

If you do not specify the **quota** argument or specify it as 0, VMS supplies a default value for each quota.

For example, in VAX MACRO you may specify a quota list, as follows:

```
QLIST:  .BYTE   PQL$_PRCLM      ; Limit number of subprocesses
        .LONG   2               ; Max = 2 subprocesses
        .BYTE   PQL$_ASTLM      ; Limit number of asts
        .LONG   6               ; Max = 6 outstanding asts
        .BYTE   PQL$_LISTEND    ; End of quota list
```

The $PQLDEF macro defines symbolic names for quotas.

Individual Quota Descriptions

A description of each quota follows. The description of each quota lists its minimum value (a SYSGEN parameter), its default value (a SYSGEN parameter), and whether it is deductible, nondeductible, or pooled. These terms have the following meaning:

Minimum value	You cannot create a process if it does not have a quota equal to or greater than this minimum. You obtain the minimum value for a quota by running SYSGEN to display the corresponding SYSGEN parameter.
Default value	If the quota list does not specify a value for a particular quota, the system assigns the process this default value. You obtain the default value by running SYSGEN to display the corresponding SYSGEN parameter.
Deductible quota	When you create a subprocess, the value for a deductible quota is subtracted from the creator's current quota and is returned to the creator when the subprocess is deleted. There is currently only one deductible quota, the CPU time limit. Note that quotas are never deducted from the creator when a detached process is created.
Nondeductible quota	Nondeductible quotas are established and maintained separately for each process and subprocess.

Pooled quota	Pooled quotas are established when a detached process is created and are shared by that process and all its descendent subprocesses. Charges against pooled quota values are subtracted from the current available totals as they are used and are added back to the total when they are not being used.

To run SYSGEN to determine the minimum and default values of a quota, enter the following sequence of commands:

```
$ RUN SYS$SYSTEM:SYSGEN
SYSGEN> SHOW/PQL
```

Minimum values are named PQL_M*xxxxx*, where *xxxxx* are the last five characters of the quota name.

Default values are named PQL_D*xxxxx*, where *xxxxx* are the last five characters of the quota name.

Individual Quotas

PQL$_ASTLM

AST limit. This quota restricts both the number of outstanding AST routines specified in system service calls that accept an AST address and the number of scheduled wakeup requests that can be issued.

Minimum: PQL_MASTLM
Default: PQL_DASTLM
Nondeductible

PQL$_BIOLM

Buffered I/O limit. This quota limits the number of outstanding system-buffered I/O operations. A buffered I/O operation is one that uses an intermediate buffer from the system pool rather than a buffer specified in a process's $QIO request.

Minimum: PQL_MBIOLM
Default: PQL_DBIOLM
Nondeductible

PQL$_BYTLM

Buffered I/O byte count quota. This quota limits the amount of system space that can be used to buffer I/O operations or to create temporary mailboxes.

Minimum: PQL_MBYTLM
Default: PQL_DBYTLM
Pooled

PQL$_CPULM

CPU time limit, specified in units of 10 milliseconds. This quota limits the total amount of CPU time that a created process can use. When it has exhausted its CPU time limit quota, the created process is deleted and the status code SS$_EXCPUTIM is returned.

If you do not specify this quota and the created process is a detached process, the detached process receives a default value of 0, that is, unlimited CPU time.

If you do not specify this quota and the created process is a subprocess, the subprocess receives half the CPU time limit quota of the creating process.

If you specify this quota as 0, the created process has unlimited CPU time, provided the creating process also has unlimited CPU time. If, however, the creating process does not have unlimited CPU time, the created process receives half the CPU time limit quota of the creating process.

The CPU time limit quota is a consumable quota; that is, the amount of CPU time used by the created process is not returned to the creating process when the created process is deleted.

Minimum: PQL_MCPULM
Default: PQL_DCPULM
Deductible

PQL$_DIOLM

Direct I/O quota. This quota limits the number of outstanding direct I/O operations. A direct I/O operation is one for which the system locks the pages containing the associated I/O buffer in memory for the duration of the I/O operation.

Minimum: PQL_MDIOLM
Default: PQL_DDIOLM
Nondeductible

PQL$_ENQLM

Lock request quota. This quota limits the number of lock requests that a process can queue.

Minimum: PQL_MENQLM
Default: PQL_DENQLM
Pooled

PQL$_FILLM

Open file quota. This quota limits the number of files that a process can have open at one time.

Minimum: PQL_MFILLM
Default: PQL_DFILLM
Pooled

PQL$_JTQUOTA

Job table quota. This quota limits the number of bytes of system paged pool used for the job logical name table. If the process being created is a subprocess, this item is ignored.

Minimum: PQL_MJTQUOA
Default: PQL_DJTQUOTA
Deductible

PQL$_PGFLQUOTA

Paging file quota. This quota limits the number of pages that can be used to provide secondary storage in the paging file for the execution of a process.

Minimum: PQL_MPGFLQUOTA
Default: PQL_DPGFLQUOTA
Pooled

PQL$_PRCLM

Subprocess quota. This quota limits the number of subprocesses a process can create.

Minimum: PQL_MPRCLM
Default: PQL_DPRCLM
Pooled

PQL$_TQELM

Timer queue entry quota. This quota limits both the number of timer queue requests a process can have outstanding and the creation of temporary common event flag clusters.

Minimum: PQL_MTQELM
Default: PQL_DTQELM
Pooled

PQL$_WSDEFAULT

Default working set size. This quota defines the number of pages in the default working set for any image the process executes. The working set size quota determines the maximum size you can specify for this quota.

Minimum: PQL_MWSDEFAULT
Default: PQL_DWSDEFAULT
Nondeductible

PQL$_WSEXTENT

Working set expansion quota. This quota limits the maximum size to which an image can expand its working set size with the Adjust Working Set Limit ($ADJWSL) system service.

Minimum: PQL_MWSEXTENT
Default: PQL_DWSEXTENT
Nondeductible

PQL$_WSQUOTA

Working set size quota. This quota limits the maximum size to which an image can lock pages in its working set with the Lock Pages in Memory ($LCKPAG) system service.

Minimum: PQL_MWSQUOTA
Default: PQL_DWSQUOTA
Nondeductible

Use of the Quota List

The values specified in the quota list are not necessarily the quotas that are actually assigned to the created process. The $CREPRC service performs the following steps to determine the quota values that are assigned:

1. It constructs a default quota list for the process being created, assigning it the default values for all quotas. Default values are SYSGEN parameters and so may vary from system to system.
2. It reads the specified quota list, if any, and updates the corresponding items in the default list. If the quota list contains multiple entries for a quota, only the last specification is used.

3 For each item in the updated quota list, it compares the quota value with the minimum value required (also a SYSGEN parameter) and uses the larger value. Then, the following occurs:

- If a subprocess is being created or a detached process is being created and the creator does not have DETACH privilege, the resulting value is compared with the current value of the corresponding quota of the creator and the lesser value is used.

  Then, if the quota is a deductible quota, that value is deducted from the creator's quota, and a check is performed to ensure that the creator will still have at least the minimum quota required. If not, the condition value SS$_EXQUOTA is returned and the subprocess or detached process is not created.

  Pooled quota values are ignored.

- If a detached process is being created and the creator has DETACH privilege, the resulting value is not compared with the current value of the corresponding quota of the creator and the resulting value is not deducted from the creator's quota. The $CREPRC service does not check that a specified quota value exceeds the maximum allowed by the system.

prcnam

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

Process name to be assigned to the created process. The **prcnam** argument is the address of a character string descriptor pointing to a 1- to 15-character process name string.

If a subprocess is being created, the process name is implicitly qualified by the UIC group number of the creating process. If a detached process is being created, the process name is qualified by the group number specified in the **uic** argument.

baspri

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

Base priority to be assigned to the created process. The **baspri** argument is a longword value in the range 0 to 31, where 31 is the highest priority and 0 is the lowest. Usual priorities are in the range 0 to 15, and real-time priorities are in the range 16 to 31.

You need the ALTPRI privilege to set a priority higher than your own. If the caller does not have this privilege, the specified base priority is compared with the caller's priority and the lower of the two values is used.

uic

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

User identification code (UIC) to be assigned to the created process. The **uic** argument is a longword value containing the UIC.

If you do not specify the **uic** argument or you specify it as 0 (the default), $CREPRC creates a process and assigns it the UIC of the creating process.

If you specify a nonzero value for the **uic** argument, $CREPRC creates a detached process. This value is interpreted as a 32-bit octal number, with two 16-bit fields:

bits 0–15—member number
bits 16–31—group number

You need the DETACH privilege to create a detached process with a UIC that is different from the UIC of the creating process.

mbxunt

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

Unit number of a mailbox to receive a termination message when the created process is deleted. The **mbxunt** argument is a word containing this number.

If you do not specify the **mbxunt** argument or specify it as 0 (the default), VMS sends no termination message when it deletes the process.

The Get Device/Volume Information ($GETDVI) service must be used to obtain the unit number of the mailbox.

If you specify the **mbxunt** argument, the mailbox is used only after the created process actually terminates. At that time, the $ASSIGN service is issued for the mailbox in the context of the terminating process and an accounting message is sent to the mailbox. If the mailbox no longer exists, cannot be assigned, or is full, the error is treated as if no mailbox had been specified.

The accounting message is sent before process rundown is initiated but after the process name has been set to null. Thus, a significant interval of time can occur between the sending of the accounting message and the final deletion of the process.

To receive the accounting message, the caller must issue a read to the mailbox. When the I/O completes, the second longword of the I/O status block, if one is specified, contains the process identification of the deleted process.

The $ACCDEF macro defines symbolic names for offsets of fields within the accounting message. The offsets, their symbolic names, and the contents of each field are shown in the following table. Unless stated otherwise, the length of the field is four bytes.

Offset	Symbolic Name	Contents
0	ACC$W_MSGTYP	MSG$_DELPROC (2 bytes)
2		Not used (2 bytes)
4	ACC$L_FINALSTS	Exit status code
8	ACC$L_PID	Process identification
12		Not used (4 bytes)
16	ACC$Q_TERMTIME	Current time in system format at process termination (8 bytes)
24	ACC$T_ACCOUNT	Account name for process, blank filled (8 bytes)
32	ACC$T_USERNAME	User name, blank filled (12 bytes)
44	ACC$L_CPUTIM	CPU time used by the process, in 10-millisecond units
48	ACC$L_PAGEFLTS	Number of page faults incurred by the process
52	ACC$L_PGFLPEAK	Peak paging file usage
56	ACC$L_WSPEAK	Peak working set size
60	ACC$L_BIOCNT	Count of buffered I/O operations performed by the process
64	ACC$L_DIOCNT	Count of direct I/O operations performed by the process
68	ACC$L_VOLUMES	Count of volumes mounted by the process
72	ACC$Q_LOGIN	Time, in system format, that process logged in (8 bytes)
80	ACC$L_OWNER	Process identification of owner

The length of the termination message is equated to the constant ACC$K_TERMLEN.

stsflg

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

Options selected for the created process. The **stsflg** argument is a longword bit vector wherein a bit corresponds to an option. Only bits 0 to 10 are used; bits 11 to 31 are reserved and must be 0.

Each option (bit) has a symbolic name, which the $PRCDEF macro defines. You construct the **stsflg** argument by performing a logical OR operation using the symbolic names of each desired option. The following table describes the symbolic name of each option:

Symbolic Name	Description
PRC$M_SSRWAIT	Disable resource wait mode.
PRC$M_SSFEXCU	Enable system service failure exception mode.
PRC$M_PSWAPM	Inhibit process swapping. PSWAPM privilege is required
PRC$M_NOACNT	Do not perform accounting. NOACNT privilege is required.
PRC$M_BATCH	Create a batch process. DETACH privilege is required.
PRC$M_HIBER	Force process to hibernate before it executes the image
PRC$M_IMGDMP	Enable image dump facility. If an image terminates due to an unhandled condition, the image dump facility writes the contents of the address space to a file in your current default directory. The file name is the same as the name of the terminated image. The file type is DMP.
PRC$M_NOUAF	Do not check authorization file if the process is detached and the image is LOGINOUT.EXE. You should not specify this option if a subprocess is being created. In previous versions of VMS, the symbolic name of this option was PRC$M_LOGIN. The symbolic name has been changed to more accurately denote the effect of setting this bit. For compatibility with existing user programs, you can still specify this bit as PRC$M_LOGIN.
PRC$M_NETWRK	Create a process that is a network connect object. DETACH privilege required.
PRC$M_DISAWS	Disable system-initiated working set adjustment.
PRC$M_DETACH	Create a detached process.
PRC$M_INTER	Create an interactive process. This option is meaningful only if the **image** argument specifies SYS$SYSTEM:LOGINOUT.EXE. The purpose of this option is to provide you with information about the process. When you specify this option, it identifies the process as one that is in communication with another user, an interactive process. For example, if you make an inquiry, using the DCL lexical function F$MODE, about a process that has specified the PRC$M_INTER option, F$MODE returns the value "INTERACTIVE."

Symbolic Name	Description
PRC$M_NOPASSWORD	Do not display the *Username:* and *Password:* prompts if the process is interactive and detached and the image is SYS$SYSTEM:LOGINOUT.EXE. If you specify this option in your call to $CREPRC, the process created by the call is logged in under the user name associated with the creating process. If you do not specify this option for an interactive process, SYS$SYSTEM:LOGINOUT.EXE prompts you for the user name and password to be associated with the process. The prompts are displayed at the SYS$INPUT device.

Note that options PRCM_BATCH, PRCM_INTER, PRCM_UAF, PRCM_NETWRK, and PRC$M_NOPASSWORD are intended for use by DIGITAL software. Complete documentation of the possible ramifications of their use is not provided.

DESCRIPTION

The calling process must have the following:

- DETACH or CMKRNL privilege to create any of the following types of process:
  - A detached process with a UIC that is different from the UIC of the calling process
  - A batch process
  - A network process
- ALTPRI privilege to create a subprocess with a higher base priority than the calling process
- SETPRV privilege to create a process with privileges that the calling process does not have
- PSWAPM privilege to create a process with process swap mode disabled
- NOACNT privilege to create a process with accounting functions disabled
- NETMBX privilege to create a network connect object

A detached process is a fully independent process. For example, the process that the system creates when you log in is a detached process.

A subprocess, on the other hand, is related to its creator in a treelike structure; it receives a portion of the creating process's resource quotas and must terminate before the creating process. The **uic** argument or the PRC$M_DETACH flag controls whether the created process is a subprocess or a detached process.

The $CREPRC service requires system dynamic memory.

The number of subprocesses that a process can create is controlled by the subprocess (PRCLM) quota; this quota is returned when a subprocess is deleted.

The number of detached processes that a process can create with the same user name is controlled by the MAXDETACH entry in the user authorization file (UAF).

When a subprocess is created, the value of any deductible quota is subtracted from the total value the creator has available; and when the subprocess is deleted, the unused portion of any deductible quota is added back to the total available to the creator. Any pooled quota value is shared by the creator and all its subprocesses.

Some error conditions are not detected until the created process executes. These conditions include an invalid or nonexistent image; invalid SYS$INPUT, SYS$OUTPUT, or SYS$ERROR logical name equivalence; inadequate quotas; or insufficient privilege to execute the requested image.

All subprocesses created by a process must terminate before the creating process can be deleted. If subprocesses exist when their creator is deleted, they are automatically deleted.

A created process is unable to run an image that calls the Run-Time Library procedure LIB$DO_COMMAND unless the process was created with the **image** argument specifying SYS$SYSTEM:LOGINOUT.EXE. This is so because SYS$SYSTEM:LOGINOUT.EXE causes a command language interpreter to be mapped into the created process, a prerequisite for calling LIB$DO_COMMAND.

A detached process is considered an interactive process only if (1) the process is created with the PRC$M_INTER option specified and (2) SYS$INPUT is not defined as a file-oriented device.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The caller cannot read a specified input string or string descriptor, the privilege list, or the quota list; or the caller cannot write the process identification.
SS$_DUPLNAM	The specified process name duplicates one already specified within that group.
SS$_EXQUOTA	At least one of the three following conditions is true: • The process has exceeded its quota for the creation of subprocesses. • A quota value specified for the creation of a subprocess exceeds the creator's corresponding quota. • The quota is deductible and the remaining quota for the creator would be less than the minimum.
SS$_INSFMEM	The system dynamic memory is insufficient for the requested operation.

SS$_IVLOGNAM	At least one of the following two conditions is true: • The specified process name has a length of 0 or has more than 15 characters. • The specified image name, input name, output name, or error name has more than 255 characters.
SS$_IVQUOTAL	The quota list is not in the proper format.
SS$_IVSTSFLG	You set a reserved status flag.
SS$_NOPRIV	The caller violated one of the privilege restrictions.
SS$_NOSLOT	No process control block is available; in other words, the maximum number of processes that can exist concurrently in the system has been reached.
SS$_INSSWAPSPACE	The swap space is insufficient for creating the process.
SS$_EXPRCLM	The creation of a detached process failed because the creating process already reached its limit for the creation of detached processes. This limit is established by the MAXDETACH quota in the user authorization file (UAF) of the creating process.

$CRETVA Create Virtual Address Space

The Create Virtual Address Space service adds a range of demand-zero allocation pages to a process's virtual address space for the execution of the current image.

FORMAT

SYS$CRETVA *inadr ,[retadr] ,[acmode]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

inadr

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

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

retadr

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

Address of a 2-longword array to receive the starting and ending virtual addresses of the pages created.

acmode

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

Access mode and protection for the new pages. 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$C_KERNEL	Kernel
PSL$C_EXEC	Executive
PSL$C_SUPER	Supervisor
PSL$C_USER	User

The most privileged access mode used is the access mode of the caller. The protection of the pages is read/write for the resultant access mode and those more privileged.

DESCRIPTION

The paging file quota (PGFLQUOTA) of the process must be sufficient to accommodate the increased size of the virtual address space.

Pages are created starting at the address contained in the first longword of the location addressed by the **inadr** argument and ending with the second longword. The ending address can be lower than the starting address. The **retadr** argument indicates the byte addresses of the pages created.

If an error occurs while pages are being created, the **retadr** argument, if specified, indicates the pages that were successfully created before the error occurred. If no pages were created, both longwords of the **retadr** argument contain a −1.

If $CRETVA creates pages that already exist, the service deletes those pages if they are not owned by a more privileged access mode than that of the caller. Any such deleted pages are reinitialized as demand-zero pages.

Note that the Expand Program/Control Region ($EXPREG) service also adds pages to a process's virtual address space.

Note: Do not use the $CRETVA system service in conjunction with other user-written procedures or DIGITAL-supplied procedures (including Run-Time Library procedures). This system service provides 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. You can find documentation on LIB$GET_VM in the *VMS Run-Time Library Routines Volume*. When using $DELTVA, you should take care to delete only pages that you have specifically created.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The **inadr** argument cannot be read by the caller, or the **retadr** argument cannot be written by the caller.
SS$_EXQUOTA	The process has exceeded its paging file quota.
SS$_INSFWSL	The process's working set limit is not large enough to accommodate the increased size of the virtual address space.

SS$_NOPRIV	A page in the specified range is in the system address space.
SS$_PAGOWNVIO	A page in the specified range already exists and cannot be deleted because it is owned by a more privileged access mode than that of the caller.
SS$_VASFULL	The process's virtual address space is full; no space is available in the page tables for the requested pages.

$CRMPSC Create and Map Section

The Create and Map Section service allows a process to associate (map) a section of its address space with (1) a specified section of a file (a disk file section) or (2) specified physical addresses represented by page frame numbers (a page frame section). This service also allows the process to create either type of section, and to specify that the section be available only to the creating process (private section) or to all processes that map to it (global section).

FORMAT

SYS$CRMPSC *[inadr] ,[retadr] ,[acmode] ,[flags] ,[gsdnam] ,[ident] ,[relpag] ,[chan] ,[pagcnt] ,[vbn] ,[prot] ,[pfc]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

inadr

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

Starting and ending virtual addresses into which the section is to be mapped. The **inadr** argument is the address of a 2-longword array containing, in order, the starting and ending process virtual addresses. Only the virtual page number portion of each virtual address is used; the low-order 9 bits are ignored.

If the starting and ending virtual addresses are the same, a single page is mapped, unless you set the SEC$M_EXPREG bit in the **flags** argument. If you set this bit, the specified address determines only whether the section is mapped in the program (P0) or control (P1) region.

If you do not specify the **inadr** argument or specify it as 0, the section is not mapped.

retadr

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

Starting and ending process virtual addresses into which the section was actually mapped by $CRMPSC. The **retadr** argument is the address of a 2-longword array containing, in order, the starting and ending process virtual addresses.

acmode

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

Access mode that is to be the owner of the pages created during the mapping. 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$C_KERNEL	Kernel
PSL$C_EXEC	Executive
PSL$C_SUPER	Supervisor
PSL$C_USER	User

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

flags

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

Flag mask specifying the type of section to be created or mapped to, as well as its characteristics. The **flags** argument is a longword bit vector wherein each bit corresponds to a flag. The $SECDEF macro defines a symbolic name for each flag. You construct the **flags** argument by performing a logical OR operation on the symbol names for all desired flags. The following table describes each flag and the default value that it supersedes:

Flag	Description
SEC$M_GBL	Pages form a global section. The default is private section.
SEC$M_CRF	Pages are copy-on-reference. By default, pages are shared.
SEC$M_DZRO	Pages are demand-zero pages. By default, they are not zeroed when copied.

Flag	Description
SEC$M_EXPREG	Pages are mapped into the first available space. By default, pages are mapped into the range specified by the **inadr** argument.
SEC$M_WRT	Pages form a read/write section. By default, pages form a read-only section.
SEC$M_PERM	Pages are permanent. By default, pages are temporary.
SEC$M_PFNMAP	Pages form a page-frame section. By default, pages form a disk-file section. Pages mapped by SEC$M_PFNMAP are not included in or charged against the process's working set; they are always valid. Do not lock these pages in the working set by using $LKWSET; this may result in a machine check if they are in I/O space.
SEC$M_SYSGBL	Pages form a system global section. By default, pages form a group global section.
SEC$M_PAGFIL	Pages form a global page-file section. By default, pages form a disk-file section.
SEC$M_EXECUTE	Pages are mapped if the caller has execute access. This flag is valid only (1) when specified from executive or kernel mode and (2) when the SEC$M_GBL flag is also specified. By default, the pages are mapped whether the caller has execute access or not.
SEC$M_NO_OVERMAP	The mapped section overmaps existing address space.

gsdnam

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

Name of the global section. The **gsdnam** argument is the address of a character string descriptor pointing to this name string.

For group global sections, VMS interprets the UIC group as part of the global section name; thus, the names of global sections are unique to UIC groups.

ident

VMS usage: **section_id**
type: **quadword (unsigned)**
access: **read only**
mechanism: **by reference**

Identification value specifying the version number of a global section and, for processes mapping to an existing global section, the criteria for matching the identification. The **ident** argument is the address of a quadword structure containing three fields.

The version number is in the second longword. The version number contains two fields: a minor identification in the low-order 24 bits and a major identification in the high-order 8 bits. You can assign values for these fields by installation convention to differentiate versions of global sections. If no version number is specified when a section is created, processes that specify a version number when mapping cannot access the global section.

The first longword specifies, in its low-order 3 bits, the matching criteria. The valid values, symbolic names by which they can be specified, and their meanings are as follows.

Value/Name	Match Criteria
0 SEC$K_MATALL	Match all versions of the section.
1 SEC$K_MATEQU	Match only if major and minor identifications match.
2 SEC$K_MATLEQ	Match if the major identifications are equal and the minor identification of the mapper is less than or equal to the minor identification of the global section.

When a section is mapped at creation time, the match control field is ignored.

If you do not specify the **ident** argument or specify it as 0 (the default), the version number and match control fields default to 0.

relpag

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

Relative page number within the global section of the first page in the section to be mapped. The **relpag** argument is a longword containing this page number.

You use this argument only for global sections. If you do not specify the **relpag** argument or specify it as 0 (the default), the global section is mapped beginning with the first virtual block in the file. This argument must be 0 for demand-zero sections in memory shared by multiple processors.

chan

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

Number of the channel on which the file has been accessed. The **chan** argument is a word containing this number.

The file must have been accessed with a VMS RMS $OPEN macro; the file options parameter (FOP) in the FAB must indicate a user file open (UFO keyword). The access mode at which the channel was opened must be the same as or less privileged than the access mode of the caller.

pagcnt

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

Number of pages in the section. The **pagcnt** argument is a longword containing this number.

The specified page count is compared with the number of pages in the section file; if they are different, the lower value is used. If you do not specify the page count or you specify it as 0 (the default), the size of the section file is used. However, for physical page frame sections, this argument must not be 0.

vbn

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

Virtual block number in the file that marks the beginning of the section. The **vbn** argument is a longword containing this number. If you do not specify the **vbn** argument or you specify it as 0 (the default), the section is created beginning with the first virtual block in the file.

If you specified page frame number mapping (by setting the SEC$M_PFNMAP flag), the **vbn** argument specifies the page frame number where the section begins in memory.

Table SYS–2 depicts which arguments are required and which are optional for three different uses of the $CRMPSC service.

Table SYS–2 Required and Optional Arguments for the $CRMPSC Service

Argument	Create/Map Global Section	Map Global[1] Section	Create/Map Private Section
inadr	Optional[2]	Required	Required
retadr	Optional	Optional	Optional
acmode	Optional	Optional	Optional

[1]The Map Global Section ($MGBLSC) service maps an existing global section.

[2]You can omit the **inadr** argument only if you want to create but not map a global section; however, in such a case, you must make the section permanent because temporary sections are automatically deleted when no processes are mapped to them. You cannot omit the **inadr** argument for demand-zero sections in memory shared by multiple processors.

Table SYS–2 (Cont.) Required and Optional Arguments for the $CRMPSC Service

Argument	Create/Map Global Section	Map Global[1] Section	Create/Map Private Section
flags			
SEC$M_GBL	Required	Ignored	Not used
SEC$M_CRF[3]	Optional	Not used	Optional
SEC$M_DZRO[3]	Optional	Not used	Optional
SEC$M_EXPREG	Optional	Optional	Optional
SEC$M_PERM	Optional[2]	Not used	Not used
SEC$M_PFNMAP	Optional	Not used	Not used
SEC$M_SYSGBL	Optional	Optional	Not used
SEC$M_WRT	Optional	Optional	Optional
SEC$M_PAGFIL	Optional	Not used	Not used
gsdnam	Required	Required	Not used
ident	Optional	Optional	Not used
relpag[3]	Optional	Optional	Not used
chan[3]	Required		Required
pagcnt	Required		Required
vbn[3]	Optional		Optional
prot	Optional		Not used
pfc[3]	Optional[4]		Optional

[1]The Map Global Section ($MGBLSC) service maps an existing global section.

[2]You can omit the **inadr** argument only if you want to create but not map a global section; however, in such a case, you must make the section permanent because temporary sections are automatically deleted when no processes are mapped to them. You cannot omit the **inadr** argument for demand-zero sections in memory shared by multiple processors.

[3]For physical page frame sections: **vbn** specifies the starting page frame number; **chan** must be zero; **relpag** and **pfc** are not used; and the SEC$M_CRF and SEC$M_DZRO flag bit settings are invalid. For page-file sections, **chan** must be zero, and **relpag** and **pfc** are not used.

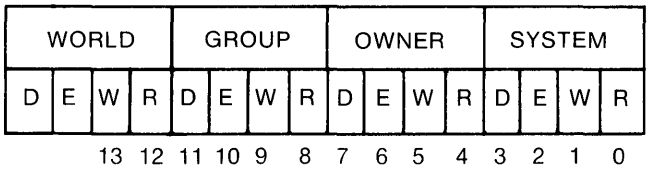
[4]This argument is not used for global sections in memory shared by multiple processors.

prot

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

Numeric value representing the protection mask to be applied to the global section. You OR this value with the protection mask associated with the file; if the file protection does not allow access to a particular category of user and the protection mask allows access, access is denied.

The mask contains four 4-bit fields. Bits are read from right to left in each field. The following diagram depicts the mask.



ZK-1706-84

Cleared bits indicate that read, write, execute, and delete access, in that order, are granted to the particular category of user.

Only read, write, and execute access are meaningful for section protection. Delete access bits are ignored. The $CRMPSC service checks the execute access bit only for calls from executive or kernel mode.

If you do not specify the **prot** argument or specify it as 0, read access and write access are granted to all users.

pfc

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

Page fault cluster size indicating how many pages are to be brought into memory when a page fault occurs for a single page. This argument is not used for page-file sections, physical page frame sections, or for global sections in memory shared by multiple processors.

DESCRIPTION

If the section pages are copy-on-reference, the process must have sufficient paging file quota (PGFLQUOTA). The systemwide number of global page-file pages is limited by the SYSGEN parameter GBLPAGFIL.

Creating a disk file section involves defining all or part of a disk file as a section. Mapping a disk file section involves making a correspondence between virtual blocks in the file and pages in the caller's virtual address space. If the $CRMPSC service specifies a global section that already exists, the service maps it.

If $CRMPSC specifies a global section and the SS$NOPRIV condition value is returned, the process may not have the required privilege to create that section. In order to create global sections, the process must have the following privileges:

- SYSGBL privilege to create a system global section
- PRMGBL privilege to create a permanent global section
- PFNMAP privilege to create a page frame section
- SHMEM privilege to create a global section in memory shared by multiple processors

Note that you do not need the PFNMAP privilege to map an existing page frame section, or the SHMEM privilege to map an existing global section in memory shared by multiple processors.

Depending on the actual operation requested, certain arguments are required or optional. Table SYS–2 summarizes how the $CRMPSC service interprets the arguments passed to it, and under what circumstances it requires or ignores arguments.

The $CRMPSC service returns the virtual addresses of the pages created in the **retadr** argument, if specified. The section is mapped from a low address to a high address, whether the section is mapped in the program or control region.

If an error occurs during the mapping of a global section, the **retadr** argument, if specified, indicates the pages that were successfully mapped when the error occurred. If no pages were mapped, both longwords of the **retadr** argument contain –1.

The SEC$M_PFNMAP flag setting identifies the memory for the section as starting at the page frame number specified in the **vbn** argument and extending for the number of pages specified in the **pagcnt** argument. Setting the SEC$M_PFNMAP flag places restrictions on the following arguments:

relpag	Does not apply
chan	Must be zero
pagcnt	Must be specified; cannot be zero
vbn	Specifies first page frame to be mapped
pfc	Does not apply

Setting the SEC$M_PFNMAP flag also places restrictions on these other flag values:

SEC$M_CRF	Must be 0
SEC$M_DZRO	Must be 0
SEC$M_PERM	Must be 1 if the flags SEC$M_GBL or SEC$M_SYSGBL are set

Setting the SEC$M_PAGFIL flag places the following restrictions on the following flags:

SEC$M_CRF	Must be 0
SEC$M_GBL	Must be 1
SEC$M_PFNMAP	Must be 0

The **flag** argument bits 4 through 13 and 18 through 31 must be 0.

The flag bit SEC$M_WRT applies only to the way in which the newly created section is mapped. For a file to be made writable, the channel used to open the file must allow write access to the file.

If the flag bit SEC$M_SYSGBL is set, the flag bit SEC$M_GBL must be set also.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully. The specified global section already exists and has been mapped.
SS$_CREATED	The service completed successfully. The specified global section did not previously exist and has been created.
SS$_ACCVIO	The **inadr** argument, **gsdnam** argument, or name descriptor cannot be read by the caller; or the **retadr** argument cannot be written by the caller.
SS$_ENDOFFILE	The starting virtual block number specified is beyond the logical end-of-file, or the value in the **relpag** argument is greater than or equal to the value in the **pagcnt** argument.
SS$_EXBYTLM	The process has exceeded the byte count quota; the system was unable to map the requested file.
SS$_EXGBLPAGFIL	The process has exceeded the system-wide limit on global page-file pages; no part of the section was mapped.
SS$_EXPORTQUOTA	The process has exceeded the number of global sections that processes on this port of the multiport (shared) memory can create.
SS$_EXQUOTA	The process exceeded its paging file quota while creating copy-on-reference or page-file-backing-store pages.
SS$_GPTFULL	There is no more room in the system global page table to set up page table entries for the section.
SS$_GSDFULL	There is no more room in the system space allocated to maintain control information for global sections.
SS$_ILLPAGCNT	The page count value is negative or is zero for a physical page frame section.
SS$_INSFMEM	Not enough pages are available in the specified shared memory to create the section.

SS$_INSFWSL	The process's working set limit is not large enough to accommodate the increased size of the address space.
SS$_INTERLOCK	The bit map lock for allocating global sections from the specified shared memory is locked by another process.
SS$_IVCHAN	An invalid channel number was specified, that is, a channel number of 0 or a number larger than the number of channels available.
SS$_IVCHNLSEC	The channel number specified is currently active.
SS$_IVLOGNAM	The specified global section name has a length of 0 or has more than 15 characters.
SS$_IVLVEC	The specified section was not installed using the /PROTECT qualifier.
SS$_IVSECFLG	An invalid flag, a reserved flag, a flag requiring a privilege you lack, or an invalid combination of flags was specified.
SS$_IVSECIDCTL	The match control field of the global section identification is invalid.
SS$_NOTFILEDEV	The device is not a file-oriented, random-access, or directory device.
SS$_NOPRIV	The process does not have the privileges to create a system global section (SYSGBL) or a permanent group global section (PRMGBL). The process does not have the privilege to create a section starting at a specific physical page frame number (PFNMAP). The process does not have the privilege to create a global section in memory shared by multiple processors (SHMEM). A page in the input address range is in the system address space. The specified channel is not assigned or was assigned from a more privileged access mode.
SS$_NOSHMBLOCK	No shared memory control block for global sections is available.
SS$_NOWRT	The section cannot be written to because the flag bit SEC$M_WRT is set, the file is read only, and the flag bit SEC$M_CRF is not set.
SS$_PAGOWNVIO	A page in the specified input address range is owned by a more privileged access mode.
SS$_SECTBLFUL	There are no entries available in the system global section table.

SS$_SHMNOTCNCT	The shared memory named in the **gsdnam** argument is not known to the system. This error can be caused by a spelling error in the string, an improperly assigned logical name, or your failure to identify the memory as shared at system generation time.
SS$_TOOMANYLNAM	The logical name translation of the **gsdnam** argument exceeded the allowed depth.
SS$_VASFULL	The process's virtual address space is full; no space is available in the page tables for the pages created to contain the mapped global section.

$DACEFC Disassociate Common Event Flag Cluster

The Disassociate Common Event Flag Cluster service releases the calling process's association with a common event flag cluster.

FORMAT

SYS$DACEFC *efn*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENT

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

Number of any event flag in the common cluster to be disassociated. The **efn** argument is a longword containing this number; however, $DACEFC uses only the low-order byte. The number must be in the range of 64 through 95 for cluster 2, and 96 through 127 for cluster 3.

DESCRIPTION

The count of processes associated with the cluster is decreased for each process that disassociates. When the image that associated with a cluster exits, the system disassociates the cluster. When the count of processes associated with a temporary cluster or with a permanent cluster that is marked for deletion reaches zero, the cluster is automatically deleted.

If a process issues this service specifying an event flag cluster with which it is not associated, the service completes successfully.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_ILLEFC	You specified an illegal event flag number. The number must be in the range of event flags 64 through 127.
SS$_INTERLOCK	The bit map lock for allocating common event flag clusters from the specified shared memory is locked by another process.

$DALLOC Deallocate Device

The Deallocate Device service deallocates a previously allocated device. The issuing process relinquishes exclusive use of the device, thus allowing other processes to assign or allocate that device.

FORMAT

SYS$DALLOC *[devnam],[acmode]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns 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 to be deallocated. The **devnam** argument is the address of a character string descriptor pointing to the device name string. The string may be either a physical device name or a logical name. If it is a logical name, it must translate to a physical device name.

If you do not specify a device name, all devices allocated by the process from access modes equal to or less privileged than that specified are deallocated.

acmode

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

Access mode from which the deallocation is to be performed. 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$C_KERNEL	Kernel
PSL$C_EXEC	Executive
PSL$C_SUPER	Supervisor
PSL$C_USER	User

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

DESCRIPTION

You can deallocate an allocated device only from access modes equal to or more privileged than the access mode from which the original allocation was made.

This service does not deallocate a device if, at the time of deallocation, the issuing process has one or more I/O channels assigned to the device; in such a case, the device remains allocated.

At image exit, the system automatically deallocates all devices that are allocated at user mode.

If you attempt to deallocate a mailbox, success is returned but no operation is performed.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The device name string or string descriptor cannot be read by the caller.
SS$_DEVASSIGN	The device cannot be deallocated because the process still has channels assigned to it.
SS$_DEVNOTALLOC	The device is not allocated to the requesting process.
SS$_IVDEVNAM	You did not specify a device name string, or the device name string contains invalid characters.
SS$_IVLOGNAM	The device name string has a length of 0 or has more than 63 characters.
SS$_NONLOCAL	The device is on a remote node.
SS$_NOPRIV	The device was allocated from a more privileged access mode.
SS$_NOSUCHDEV	The specified device does not exist in the host system.

$DASSGN Deassign I/O Channel

The Deassign I/O Channel service deassigns (releases) an I/O channel that it acquired using the Assign I/O Channel ($ASSIGN) service.

FORMAT

SYS$DASSGN *chan*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENT

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

Number of the I/O channel to be deassigned. The **chan** argument is a word containing this number.

DESCRIPTION

You can deassign an I/O channel only from an access mode equal to or more privileged than the access mode from which the original channel assignment was made.

When you deassign a channel, any outstanding I/O requests on the channel are canceled. If a file is open on the specified channel, the file is closed.

If a mailbox was associated with the device when the channel was assigned, the link to the mailbox is cleared.

If the I/O channel was assigned for a network operation, the network link is disconnected.

If the specified channel is the last channel assigned to a device that has been marked for dismounting, the device is dismounted.

I/O channels assigned from user mode are automatically deassigned at image exit.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_IVCHAN	You specified an invalid channel number, 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.

$DCLAST Declare AST

The Declare AST service queues an AST for the calling access mode or for a less privileged access mode.

FORMAT

SYS$DCLAST *astadr ,[astprm] ,[acmode]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

astadr

VMS usage: **ast_procedure**
type: **procedure entry mask**
access: **call without stack unwinding**
mechanism: **by reference**

AST service routine to be executed. The **astadr** argument is the address of the entry mask of this routine.

astprm

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

AST parameter to be passed to the AST routine specified by the **astadr** argument. The **astprm** argument is a longword containing this parameter.

acmode

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

Access mode for which the AST is to be declared. The most privileged access mode used is the access mode of the caller. The resultant mode is the access mode for which the AST is declared.

DESCRIPTION

The Declare AST service queues an AST for the calling access mode or for a less privileged access mode. For example, a routine executing in supervisor mode can declare an AST for either supervisor or user mode.

The $DCLAST service requires system dynamic memory and uses the AST limit (ASTLM) quota of the process.

The service does not validate the address of the AST service routine. If you specify an illegal address (such as 0), an access violation occurs when the AST service routine is given control.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_EXQUOTA	The process has exceeded its AST limit (ASTLM) quota.
SS$_INSFMEM	The system dynamic memory is insufficient for completing the service.

$DCLCMH Declare Change Mode or Compatibility Mode Handler

The Declare Change Mode or Compatibility Mode Handler service specifies the address of a routine to receive control when (1) a Change Mode to User or Change Mode to Supervisor instruction trap occurs, or (2) a compatibility mode fault occurs.

FORMAT

SYS$DCLCMH *addres ,[prvhnd] ,[type]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

addres

VMS usage: **procedure**
type: **procedure entry mask**
access: **call without stack unwinding**
mechanism: **by reference**

Routine to receive control when a change mode trap or a compatibility mode fault occurs. The **addres** argument is the address of a subroutine called with a JSB instruction.

If you specify the **addres** argument as 0, $DCLCMH clears the previously declared handler.

prvhnd

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

Address of a previously declared handler. The **prvhnd** argument is the address of a longword containing the address of the previously declared handler.

type

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

Handler type indicator. The **type** argument is a longword value. The value 0 (the default) indicates that a change mode handler is to be declared for

the access mode at which the request is issued; the value 1 specifies that a compatibility mode handler is to be declared.

DESCRIPTION

A change mode handler provides users with a dispatching mechanism similar to that used for system service calls. It allows a routine that executes in supervisor mode to be called from user mode. You declare the change mode handler from supervisor mode; then when the process executing in user mode issues a Change Mode to Supervisor instruction, the change mode handler receives control and executes in supervisor mode.

The operating system uses compatibility mode handlers to bypass normal condition handling procedures when an image executing in compatibility mode causes a compatibility mode exception.

Before the change mode handler exits, it must push the saved PC and PSL onto the stack from the location CTL$AL_CMCNTX, and it must exit by issuing an REI instruction.

You can declare a change mode handler only from user or supervisor mode.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The longword to receive the address of the previous change mode handler cannot be written by the caller.

$DCLEXH Declare Exit Handler

The Declare Exit Handler service declares an exit handling routine that receives control when an image exits. Image exit normally occurs when the image currently executing in a process returns control to the operating system. Image exit may also occur when you call the Exit ($EXIT) or Force Exit ($FORCEX) service.

FORMAT

SYS$DCLEXH *desblk*

RETURNS

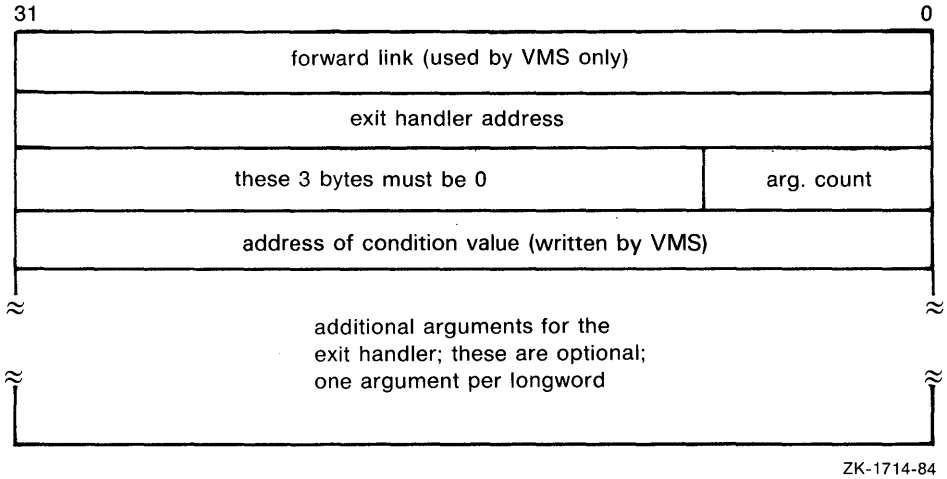
VMS usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENT

desblk
VMS usage: **exit_handler_block**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Exit handler control block. The **desblk** argument is the address of this control block. This control block, which describes the exit handler, is depicted in the following diagram.



ZK-1714-84

DESCRIPTION

Exit handlers are described by exit control blocks. The operating system maintains a separate list of these control blocks for user, supervisor, and executive modes. The $DCLEXH service adds the description of an exit handler to the front of one of these lists. The actual list to which the exit control block is added is determined by the access mode of the caller.

You can call this service only from user, supervisor, and executive modes.

At image exit, the exit handlers declared from user mode are called first; they are called in the reverse order from which they were declared.

Each exit handler is executed only once; it must be redeclared before it can be executed again. The exit handling routine is called as a normal procedure with the argument list specified in the third through nth longwords of the exit control block. The first argument is the address of a longword to receive a system status code indicating the reason for exit; the system always fills in this longword before calling the exit handler.

The Cancel Exit Handler ($CANEXH) service removes an exit control block from the list.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The first longword of the exit control block cannot be written by the caller.
SS$_IVSSRQ	The call to the service is invalid because it was made from kernel mode.
SS$_NOHANDLER	The exit handler control block address was not specified or was specified as 0.

$DELLNM Delete Logical Name

The Delete Logical Name service deletes all logical names with the specified name at the specified access mode or outer access mode, or it deletes all the logical names with the specified access mode or outer access mode in a specified table. If any logical names being deleted are also the names of logical name tables, then all of the logical names contained within those tables and all of their subtables are also deleted.

FORMAT

SYS$DELLNM *tabnam ,[lognam] ,[acmode]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

tabnam

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

Name of a logical name table or a list of tables to be searched for the logical name to be deleted. The **tabnam** argument is the address of a descriptor that points to the table name. This argument is required.

If **tabnam** is not the name of a logical name table, it is assumed to be a logical name and is translated iteratively until either the name of a logical name table is found or the number of translations allowed by the system has been performed.

If **tabnam** translates to the name of a list of tables, $DELLNM does the following:

- If you specify the **lognam** argument, $DELLNM searches (in order) each table in the list until it finds the first table that contains the specified logical name. If the logical name is at the specified access mode, $DELLNM then deletes occurrences of the logical name at the specified access mode and at outer access modes within the table.
- If you do not specify the **lognam** argument, $DELLNM deletes all of the logical names at the specified access mode or at outer access modes from the first table in the list whose access mode is equal to or less privileged than the caller's access mode.

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 descriptor that points to the logical name string.

acmode

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

Access mode to be used in the delete operation. The **acmode** argument is the address of a byte containing this access mode. The $PSLDEF macro defines symbolic names for the four access modes.

You determine the access mode actually used in the delete operation by "maximizing" the access mode of the caller with the access mode specified by the **acmode** argument; that is, the less privileged of the two is used.

However, if you have SYSNAM privilege, the delete operation is executed at the specified access mode regardless of the caller's access mode.

If you omit this argument or you specify it as 0, the access mode of the caller is used in the delete operation. The access mode used in the delete operation determines which tables are used and which names are deleted.

DESCRIPTION

Depending on the operation, the calling process may need a certain privilege to use $DELLNM:

- You need write access to the logical name table that contains a logical name to delete the logical name from a shareable table.
- You need either delete access to the logical name table or write access to the directory table that contains the table name to delete a shareable logical name table.
- You need SYSNAM privilege to delete either a logical name or table at an inner access mode.
- You need GRPNAM or SYSPRV privilege to delete a logical name from a group table.
- You need SYSNAM or SYSPRV privilege to delete a logical name from a system table.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The service cannot access the location(s) specified by one or more arguments.
SS$_BADPARAM	One or more arguments have an invalid value, or a logical name table name was not specified.

SS$_IVLOGNAM	The **lognam** argument specifies a string whose length is not in the required range of 1 through 255 characters.
SS$_IVLOGTAB	The **tabnam** argument does not specify a logical name table.
SS$_NOLOGNAM	The specified logical name table does not exist, or a logical name with an access mode equal to or less privileged than the caller's access mode does not exist in the logical name table.
SS$_NOLOGTAB	The specified logical name table does not exist.
SS$_NOPRIV	The caller lacks the necessary privilege to delete the logical name.
SS$_TOOMANYLNAM	The logical name translation of the table name exceeded the allowable depth (10 translations).

$DELMBX Delete Mailbox

The Delete Mailbox service marks a permanent mailbox for deletion. The actual deletion of the mailbox and of its associated logical name assignment occurs when no more I/O channels are assigned to the mailbox.

FORMAT

SYS$DELMBX *chan*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENT

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

Number of the channel assigned to the mailbox that is to be deleted. The **chan** argument is a word containing this number.

DESCRIPTION

You need PRMMBX privilege to delete a permanent mailbox.

You can delete a mailbox only from an access mode equal to or more privileged than the access mode from which the mailbox channel was assigned.

Temporary mailboxes are automatically deleted when their reference count goes to zero.

The $DELMBX service does not deassign the channel assigned by the caller, if any. The caller must deassign the channel with the Deassign I/O Channel ($DASSGN) service.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_DEVNOTMBX	The specified channel is not assigned to a mailbox.
SS$_INTERLOCK	The bit map lock for allocating mailboxes from the specified shared memory is locked by another process.

SS$_IVCHAN	You specified an invalid channel number, that is, a channel number of 0 or a number larger than the number of channels available.
SS$_NOPRIV	The specified channel is not assigned to a device; the process does not have the privilege to delete a permanent mailbox or a mailbox in memory shared by multiple processors; or the access mode of the caller is less privileged than the access mode from which the channel was assigned.

$DELPRC Delete Process

The Delete Process service allows a process to delete itself or another process.

FORMAT

SYS$DELPRC *[pidadr] ,[prcnam]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

pidadr

VMS usage: **process_id**
type: **longword (unsigned)**
access: **modify**
mechanism: **by reference**

Process identification (PID) of the process to be deleted. The **pidadr** argument is the address of a longword that contains the PID.

You must specify the **pidadr** argument to delete processes in other UIC groups.

prcnam

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

Process name of the process to be deleted. The **prcnam** is the address of a character string descriptor pointing to a 1- to 15-character process name string.

You use the **prcnam** argument to delete only processes in the same UIC group as the calling process, because process names are unique to UIC groups, and VMS uses the UIC group number of the calling process to interpret the process name specified by the **prcnam** argument.

You must use the **pidadr** argument to delete processes in other groups.

If you specify neither the **pidadr** nor **prcnam** argument, $DELPRC deletes the calling process; control is not returned.

DESCRIPTION

Depending on the operation, the calling process may need certain privileges to use $DELPRC:

- You need GROUP privilege to delete processes in the same group that do not have the same UIC.
- You need WORLD privilege to delete any process in the system.

The Delete Process system service requires system dynamic memory.

Deductible resource quotas granted to subprocesses are returned to the creator when the subprocesses are deleted.

When you delete a process or subprocess, a termination message is sent to its creator, provided the mailbox to receive the message still exists and the creating process has access to the mailbox. The termination message is sent before the final rundown is initiated; thus, the creator may receive the message before the process deletion is complete.

Due to the complexity of the required rundown operations, a significant time interval occurs between a delete request and the actual deletion of the process. However, the $DELPRC service returns to the caller immediately after initiating the rundown operation.

If you issue subsequent delete requests for a process currently being deleted, the requests return immediately with a successful completion status.

For a complete list of the actions performed by the system when it deletes a process, see the *Introduction to VMS System Services.*

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The process name string or string descriptor cannot be read by the caller, or the process identification cannot be written by the caller.
SS$_INSFMEM	The system dynamic memory is insufficient for completing the operation.
SS$_NONEXPR	The specified process does not exist, or an invalid process identification was specified.
SS$_NOPRIV	The caller does not have the privilege to delete the specified process.

$DELTVA Delete Virtual Address Space

The Delete Virtual Address Space service deletes a range of addresses from a process's virtual address space. Upon successful completion of the service, the deleted pages are inaccessible, and references to them cause access violations.

FORMAT

SYS$DELTVA *inadr ,[retadr] ,[acmode]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

inadr

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

Starting and ending virtual addresses of the pages to be deleted. The **inadr** argument is the address of a 2-longword array containing, in order, the starting and the ending process virtual addresses. If the starting and ending virtual addresses are the same, a single page is deleted. Only the virtual page number portion of each virtual address is used; the low-order 9 bits are ignored.

The $DELTVA service deletes pages starting at the address contained in the second longword of the **inadr** argument and ending at the address in the first longword. Thus, if you use the same address array for both the Create Virtual Address Space ($CRETVA) and the $DELTVA services, the pages are deleted in the reverse order from which they were created.

retadr

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

Starting and ending process virtual addresses of the pages that $DELTVA has deleted. The **retadr** argument is the address of a 2-longword array containing, in order, the starting and ending process virtual addresses.

acmode

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

Access mode on behalf of which the service is to be performed. The **acmode** argument is a longword containing the access mode.

The most privileged access mode used is the access mode of the caller. The calling process can delete pages only if those pages are owned by an access mode equal to or less privileged than the access mode of the calling process.

DESCRIPTION

If any of the pages in the specified range have already been deleted or do not exist, the service continues as if the pages were successfully deleted.

If an error occurs while pages are being deleted, the **retadr** argument specifies the pages that were successfully deleted before the error occurred. If no pages are deleted, both longwords in the return address array contain the value –1.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The input address array cannot be read by the caller, or the return address array cannot be written by the caller.
SS$_NOPRIV	A page in the specified range is in the system address space.
SS$_PAGOWNVIO	A page in the specified range is owned by an access mode more privileged than the access mode of the caller.

$DEQ Dequeue Lock Request

The Dequeue Lock Request service dequeues (unlocks) granted locks; dequeues the sublocks of a lock; or cancels an ungranted lock request. The calling process must have previously acquired the lock or queued the lock request by calling the Enqueue Lock Request ($ENQ) service.

The $DEQ, $ENQ, $ENQW, and $GETLKI services together provide the user interface to the VMS lock management facility. For additional information about lock management, refer to the descriptions of these other services and to the *Introduction to VMS System Services*.

FORMAT

SYS$DEQ *[lkid] ,[valblk] ,[acmode] ,[flags]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

lkid

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

Lock identification of the lock to be dequeued. The **lkid** argument specifies this lock identification.

Note that if you do not specify the **lkid** argument, you must specify the LCK$M_DEQALL flag in the **flags** argument.

When you specify the LCK$M_DEQALL flag in the **flags** argument, different values (or no value) for the **lkid** argument produce varying behavior:

- When you do not specify the **lkid** argument (or specify it as 0) and you do specify the LCK$M_DEQALL flag, $DEQ dequeues all locks held by the process, at access modes equal to or less privileged than the effective access mode, on all resources. The effective access mode is the least privileged of the caller's access mode and the access mode specified in the **acmode** argument.
- When you specify the **lkid** argument as a nonzero value together with the LCK$M_DEQALL flag, $DEQ dequeues all sublocks of the lock identified by **lkid**; it does not dequeue the lock identified by **lkid**. For this operation, $DEQ ignores the LCK$M_CANCEL flag if it is set. A sublock of a lock is a lock that was created when the **parid** argument in the call to $ENQ was specified, where **parid** is the lock ID of the parent lock.

If you omit the **lkid** argument (or specify it as 0) and the LCK$M_DEQALL flag is not set, the $DEQ service returns the invalid lock ID condition value (SS$_IVLOCKID).

valblk

VMS usage: **lock_value_block**
type: **longword (unsigned)**
access: **modify**
mechanism: **by reference**

Lock value block for the resource associated with the lock to be dequeued. The **valblk** argument is the address of the 16-byte lock value block. When you specify the LCK$M_DEQALL flag, you cannot use this argument.

When a protected write (PW) or exclusive (EX) mode lock is being dequeued and you specify a lock value block in the **valblk** argument, the contents of that lock value block are written to the lock value block in the lock database. Further, if the lock value block in the lock database was marked as invalid, that condition is cleared; the block becomes valid.

acmode

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

Access mode of the lock to be dequeued. 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$C_KERNEL	Kernel
PSL$C_EXEC	Executive
PSL$C_SUPER	Supervisor
PSL$C_USER	User

The effective access mode used by $DEQ is the least privileged of the caller's access mode and the access mode specified by the **acmode** argument. If you do not specify the **acmode** argument, $DEQ uses the caller's access mode.

flags

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

Flags specifying options for the $DEQ operation. The **flags** argument is a longword bit mask that is the logical OR of each bit set, where each bit corresponds to an option.

Note that if you do not specify the **lkid** argument, you must specify the LCK$M_DEQALL flag in the **flags** argument.

A symbolic name for each flag bit is defined by the $LCKDEF macro. The following table describes each flag.

Flag	Description
LCK$M_DEQALL	When you specify this flag, $DEQ dequeues multiple locks, depending on the value of the **lkid** argument. Refer to the description of the **lkid** argument for details. If you specify LCK$M_DEQALL, the LCK$M_CANCEL flag, if set, is ignored.
LCK$M_CANCEL	When you specify this flag, $DEQ attempts to cancel a lock conversion request that was queued by $ENQ. This attempt can succeed only if the lock request has not yet been granted, in which case the request is in the conversion queue. If you specify the LCK$M_DEQALL flag, the LCK$M_CANCEL flag is ignored. Specifying the LCK$M_CANCEL flag can result in the following actions: If the lock conversion request has already been granted, then the attempt to cancel the request has failed; in this case $DEQ returns the condition value SS$_CANCELGRANT in R0. If the lock conversion request has not yet been granted, $DEQ cancels the request. As a result, the lock is regranted at its previous lock mode; the $ENQ service receives the condition value SS$_CANCEL in the lock status block; and the $DEQ service returns the condition value SS$_NORMAL in R0. If the lock request was not a conversion request, but was a new lock request and was therefore on the waiting queue, $DEQ aborts the lock request. As a result, the $ENQ service receives the condition value SS$_ABORT in the lock status block, and $DEQ returns the condition value SS$_NORMAL in R0.
LCK$M_INVVALBLK	When you specify this flag, $DEQ marks the lock value block, which is maintained for the resource in the lock database, as invalid. The lock value block remains marked as invalid until it is again written to. The Description section of the $ENQ service provides additional information about lock value block invalidation. This flag is ignored if (1) the lock mode of the lock being dequeued is not protected write or exclusive, or (2) you specify the LCK$M_CANCEL flag.

DESCRIPTION

When a protected write (PW) or exclusive (EX) mode lock is being dequeued and you specify a lock value block in the **valblk** argument, the contents of that lock value block are written to the lock value block in the lock database.

If you specify the LCK$M_INVVALBLK flag in the **flags** argument and the lock mode of the lock being dequeued is PW or EX, the lock value block in the lock database is marked as invalid whether or not a lock value block was specified in the **valblk** argument.

CONDITION VALUES RETURNED

SS$_NORMAL	The lock was dequeued successfully.
SS$_ACCVIO	The value block specified by the **valblk** argument cannot be accessed by the caller.
SS$_IVLOCKID	An invalid or nonexistent lock identification was specified or the process does not have the privilege to dequeue a lock at the specified access mode.
SS$_SUBLOCKS	The lock has sublocks and cannot be dequeued.
SS$_CANCELGRANT	The LCK$M_CANCEL flag in the **flags** argument was specified, but the lock request that $DEQ was to cancel had already been granted.

$DGBLSC Delete Global Section

The Delete Global Section service marks an existing permanent global section for deletion. The actual deletion of the global section takes place when all processes that have mapped the global section have deleted the mapped pages.

FORMAT

SYS$DGBLSC *[flags] ,gsdnam ,[ident]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

flags

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

Mask indicating global section characteristics. The **flags** argument is a longword value. A value of 0 (the default) specifies a group global section; a value of SEC$M_SYSGBL specifies a system global section.

gsdnam

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

Name of the global section to be deleted. The **gsdnam** argument is the address of a character string descriptor pointing to this name string.

For group global sections, VMS interprets the group UIC as part of the global section name; thus, the names of global sections are unique to UIC groups.

ident

VMS usage: **section_id**
type: **quadword (unsigned)**
access: **read only**
mechanism: **by reference**

Identification value specifying the version number of the global section to be deleted and the matching criteria to be applied. The **ident** argument is the address of a quadword structure containing three fields.

The version number is in the second longword. The version number contains two fields: a minor identification in the low-order 24 bits and a major identification in the high-order 8 bits. Values for these fields can be assigned by installation convention to differentiate versions of global sections. If you specify no version number when creating a section, processes that specify a version number when mapping cannot access the global section.

The first longword specifies, in its low-order 3 bits, the matching criteria. The valid values, symbolic names by which they can be specified, and their meanings are listed in the following table.

Value	Name	Match Criteria
0	SEC$K_MATALL	Match all versions of the section
1	SEC$K_MATEQU	Match only if major and minor identifications match
2	SEC$K_MATLEQ	Match if the major identifications are equal and the minor identification of the mapper is less than or equal to the minor identification of the global section

If you specify no address or specify it as 0 (the default), the version number and match control fields default to 0.

DESCRIPTION

Depending on the operation, the calling process may need a certain privilege to use $DGBLSC:

- You need SYSGBL privilege to delete a system global section.
- You need PRMGBL privilege to delete a permanent global section.
- You need PFNMAP privilege to delete a page frame section.
- You need SHMEM privilege to delete a global section located in memory shared by multiple processors.

After a global section has been marked for deletion, any process that attempts to map it receives the warning return status code SS$_NOSUCHSEC.

Temporary global sections are automatically deleted when the count of processes using the section goes to 0.

The $DGBLSC service does not unmap a global section from a process's virtual address space. To do this, the process should call the Delete Virtual Address Space ($DELTVA) service, which deletes the pages to which the section is mapped.

A section located in memory that is shared by multiple processors can be marked for deletion only by a process running on the same processor that created the section.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The global section name or name descriptor or the section identification field cannot be read by the caller.
SS$_INTERLOCK	The bit map lock for allocating global sections from the specified shared memory is locked by another process.
SS$_IVLOGNAM	The global section name has a length of 0 or has more than 15 characters.
SS$_IVSECFLG	You set an invalid flag, reserved flag, or flag requiring a user privilege.
SS$_IVSECIDCTL	The section identification match control field is invalid.
SS$_NOPRIV	The caller does not have the privilege to delete a system global section, does not have read/write access to a group global section, or does not have the privilege to delete a global section located in memory that is shared by multiple processors.
SS$_NOSUCHSEC	The specified global section does not exist, or the identifications do not match.
SS$_NOTCREATOR	The section is in memory shared by multiple processors and was created by a process on another processor.
SS$_SHMNOTCNCT	The shared memory named in the **gsdnam** argument is not known to the system. This error can be caused by a spelling error in the string, an improperly assigned logical name, or the failure to identify the memory as shared at system generation time.
SS$_TOOMANYLNAM	The logical name translation of the gsdnam string exceeded the allowed depth of 10.

$DISMOU Dismount Volume

The Dismount Volume service dismounts a mounted volume or volume sets.

FORMAT

SYS$DISMOU *devnam ,[flags]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns 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**

Device name of the device to be dismounted. The **devnam** argument is the address of a character string descriptor pointing to the device name string. The string may be either a physical device name or a logical name. If it is a logical name, it must translate to a physical device name.

flags

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

A longword bit vector specifying options for the dismount operation. The **flags** argument is a longword bit vector wherein a bit, when set, selects the corresponding option. Each bit has a symbolic name; these names are defined by the $DMTDEF macro. The flags and their meanings are listed in the following table.

Flag	Meaning
DMT$M_ABORT	The volume is to be dismounted even if the caller did not mount the volume. If the volume was mounted with MNT$M_SHARE specified, $DISMOU dismounts the volume for all of the users who mounted it. To specify DMT$M_ABORT, the caller must: (1) have GRPNAM privilege for a group volume; (2) have SYSNAM privilege for a system volume; or (3) either own the volume or have VOLPRO privilege.
DMT$M_CLUSTER	The volume is to be dismounted cluster wide, that is, from all nodes in the VAXcluster. $DISMOU dismounts the volume from the caller's node first, and then from every other node in the existing VAXcluster. DMT$M_CLUSTER dismounts only system or group volumes. To dismount a group volume clusterwide, the caller must have GRPNAM privilege. To dismount a system volume cluster wide, the caller must have SYSNAM privilege. DMT$M_CLUSTER has no effect if the system is not a member of a VAXcluster. DMT$M_CLUSTER applies only to disks.
DMT$M_NOUNLOAD	Volume is not unloaded.
DMT$M_UNIT	The specified device, rather than the entire volume set, is dismounted.

DESCRIPTION

Depending on the operation, the calling process may need a certain privilege to use $DISMOU:

- You need GRPNAM privilege to dismount a volume mounted with the /GROUP qualifier.
- You need SYSNAM privilege to dismount a volume mounted with the /SYSTEM qualifier.

To dismount a private volume, the caller must own the volume.

When you issue the $DISMOU service, $DISMOU removes the volume from your list of mounted volumes, deletes the logical name (if any) associated with the volume, and decrements the mount count.

If the mount count does not equal 0 after being decremented, $DISMOU does not mark the volume for dismounting (because the volume must have been mounted shared). In this case, the total effect for the issuing process is that the process is denied access to the volume and a logical name entry is deleted.

If the mount count equals 0 after being decremented, $DISMOU marks the volume for dismounting. After marking the volume for dismounting, $DISMOU waits until the volume is idle before dismounting it. A native volume is idle when no user has an open file to the volume, and a foreign volume is idle when no channels are assigned to the volume.

Native volumes are Files–11 structured disks or ANSI-structured tapes. Foreign volumes are not Files–11 or ANSI structured.

After a volume is dismounted, nonpaged pool is returned to the system. Paged pool is also returned if you mounted the volume using the /GROUP or /SYSTEM qualifier.

If a volume is part of a Files–11 volume set and the flag bit DMT$V_UNIT is not set, the entire volume set is dismounted.

When a Files–11 volume has been marked for dismount, new channels can be assigned to the volume, but no new files can be opened.

Note that the SS$_NORMAL status code indicates only that $DISMOU has successfully performed one or more of the actions just described: decremented the mount count, marked the volume for dismount, or dismounted the volume. The only way to determine that the dismount has actually occurred is to check the device characteristics using the Get Device/Volume Information ($GETDVI) service.

By specifying the DVI$_DEVCHAR item code in a call to $GETDVI, you can learn whether a volume is mounted (it is if the DEV$V_MNT bit is set) or whether it is marked for dismounting (it is if the DEV$M_DMT bit is set). If DEV$V_MNT is clear or if DEV$M_DMT is set, the mount count is zero.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The device name descriptor cannot be read or does not describe a readable device name.
SS$_DEVALLOC	The device is allocated to another process and cannot be dismounted by the caller.
SS$_IVLOGNAM	The device logical name has a length of zero or is longer than the allowable logical name length.
SS$_IVDEVNAM	The device name string is not valid.
SS$_NOGRPNAM	The GRPNAM privilege is required to dismount a volume mounted for groupwide access.
SS$_NOIOCHAN	No I/O channel is available. To use $DISMOU, a channel must be assigned to the volume.
SS$_NONLOCAL	The device is on a remote node.
SS$_NOSUCHDEV	The specified device does not exist.
SS$_NOSYSNAM	The SYSNAM privilege is required to dismount a volume mounted for systemwide access.
SS$_NOTFILEDEV	The specified device is not file-structured.
SS$_DEVOFFLINE	The specified device is not available.
SS$_DEVNOTMOUNT	The specified device is not mounted.

$DLCEFC Delete Common Event Flag Cluster

The Delete Common Event Flag Cluster service marks a permanent common event flag cluster for deletion. The cluster is actually deleted when no more processes are associated with it.

FORMAT

SYS$DLCEFC *name*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENT

name
VMS usage: **ef_cluster_name**
type: **character-coded text string**
access: **read only**
mechanism: **by descriptor—fixed-length string descriptor**

Name of the common event flag cluster to be deleted. The **name** argument is the address of a character string descriptor pointing to the name of the cluster.

The names of event flag clusters are unique to UIC groups, and the UIC group number of the calling process is part of the name. Refer to the *Introduction to VMS System Services* for more information on this argument.

DESCRIPTION

To delete a common event flag cluster, the calling process must either have PRMCEB privilege or have the same UIC as the process that created the cluster.

The $DLCEFC service does not disassociate a process from a common event flag cluster; the Disassociate Common Event Flag Cluster ($DACEFC) service does this. However, the system disassociates a process from an event flag cluster at image exit.

If the cluster has already been deleted or does not exist, the $DLCEFC service returns the status code SS$_NORMAL.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_IVLOGNAM	The cluster name string has a length of 0 or has more than 15 characters.
SS$_NOPRIV	The process does not have the privilege to delete a permanent common event flag cluster, or the process does not have the privilege to delete a common event flag cluster in memory shared by multiple processors.

$ENQ Enqueue Lock Request

The Enqueue Lock Request service queues a new lock or lock conversion on a resource. The $ENQ service completes asynchronously; that is, it returns to the caller after queueing the lock request, without waiting for the lock to be either granted or converted. For synchronous completion, use the Enqueue Lock Request and Wait ($ENQW) service. The $ENQW service is identical to the $ENQ service in every way except that $ENQW returns to the caller when the lock is either granted or converted.

For additional information about system service completion, refer to the Synchronize ($SYNCH) service and to the *Introduction to VMS System Services*.

The $ENQ, $ENQW, $DEQ (Dequeue Lock Request), and $GETLKI (Get Lock Information) services together provide the user interface to the VMS lock management facility. Refer to the descriptions of these other services and to the *Introduction to VMS System Services* for additional information about lock management.

FORMAT

SYS$ENQ *[efn] ,lkmode ,lksb [,flags] [,resnam] [,parid] [,astadr] [,astprm] [,blkast] [,acmode] [,nullarg]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

efn

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

Number of the event flag to be set when the lock request has been granted. The **efn** argument is a longword containing this number; however, $ENQ uses only the low-order byte.

Upon request initiation, $ENQ clears the specified event flag (or event flag 0 if **efn** was not specified). Then, when the lock request is granted, the specified event flag (or event flag 0) is set unless you specified the LCK$M_SYNCSTS flag in the **flags** argument.

lkmode

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

Lock mode requested. The **lkmode** argument is a longword specifying this lock mode.

Each lock mode has a symbolic name. The $LCKDEF macro defines these symbolic names. The following table gives the symbolic name and description for each lock mode:

Lock Mode	Description
LCK$K_NLMODE	Null mode. This mode grants no access to the resource but serves rather as a place holder and indicator of future interest in the resource. The null mode does not inhibit locking at other lock modes; further, it prevents the deletion of the resource and lock value block, which would otherwise occur if the locks held at the other lock modes were dequeued.
LCK$K_CRMODE	Concurrent read. This mode grants the caller read access to the resource while permitting write access to the resource by other users. This mode is used to read data from a resource in an unprotected manner, because other users can modify that data as it is being read. This mode is typically used when additional locking is being performed at a finer granularity with sublocks.
LCK$K_CWMODE	Concurrent write. This mode grants the caller write access to the resource while permitting write access to the resource by other users. This mode is used to write data to a resource in an unprotected fashion, because other users may simultaneously write data to the resource. This mode is typically used when additional locking is being performed at a finer granularity with sublocks.
LCK$K_PRMODE	Protected read. This mode grants the caller read access to the resource while permitting only read access to the resource by other users. Write access is not allowed. This is the traditional "share lock."
LCK$K_PWMODE	Protected write. This mode grants the caller write access to the resource while permitting only read access to the resource by other users; the other users must have specified concurrent read mode access. No other writers are allowed access to the resource. This is the traditional "update lock."
LCK$K_EXMODE	Exclusive. The exclusive mode grants the caller write access to the resource and allows no access to the resource by other users. This is the traditional "exclusive lock."

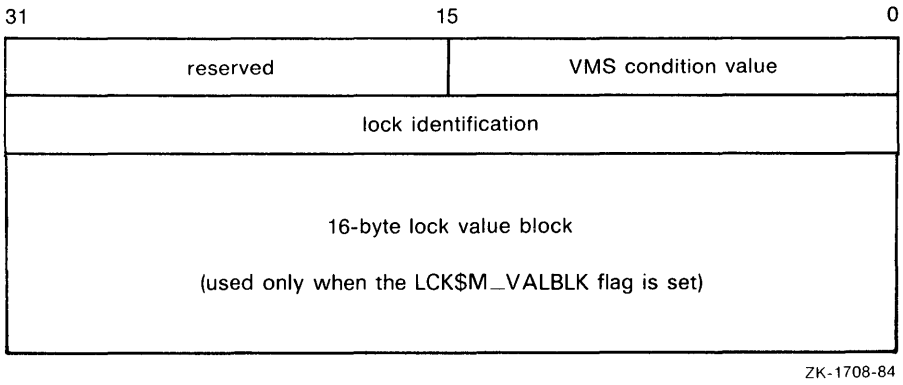
lksb

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

Lock status block in which $ENQ writes the final completion status of the operation. The **lksb** argument is the address of the 8-byte lock status block.

The lock status block may optionally contain a 16-byte lock value block. When you specify the LCK$M_VALBLK flag in the **flags** argument, the lock status block contains a lock value block; in this case, the 16-byte lock value block appears beginning at the first byte following the eighth byte of the lock status block, bringing the total length of the lock status block to 24 bytes.

The following diagram shows the format of the lock status block and the optional lock value block.



ZK-1708-84

Lock Status Block Fields

Condition value

A word in which $ENQ writes a VMS condition value describing the final disposition of the lock request, for example, whether the lock was granted, converted, and so on. The condition values returned in this field are described under the heading "Condition Values Returned in the Lock Status Block," which appears following the list of condition values returned in R0.

Reserved

A word reserved by DIGITAL.

Lock identification

A longword containing the lock identification of the lock.

For a new lock, $ENQ writes the lock identification of the requested lock into this longword when the lock request is queued.

For a lock conversion on an existing lock, you must supply the lock identification of the existing lock in this field.

Lock value block

A user-defined, 16-byte structure containing information about the resource. This information is user defined and is interpreted only by the user program.

When a process acquires a lock on a resource, the lock management facility provides that process with a process-private copy of the lock value block associated with the resource, provided that process has specified the LCK$M_VALBLK flag in the **flags** argument. The copy provided to the process is a copy of the lock value block stored in the lock manager's database.

The copy of the lock value block maintained in the lock database is updated in the following way: whenever a process either (1) dequeues a lock at protected write (PW) or exclusive (EX) mode or (2) converts a lock at one of these modes to a lower lock mode, VMS stores the caller's lock value block in the lock database, provided the caller has specified the LCK$M_VALBLK flag.

Callers of $ENQ are provided with copies of the updated lock value block from the lock database in the following way: when $ENQ grants a new lock to the caller or converts the caller's existing lock to a higher lock mode, $ENQ copies the lock value block from the lock database to the caller's lock value block, provided the caller has specified the LCK$M_VALBLK flag.

The Description section describes events that may cause the lock value block to become invalid.

flags

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

Flags specifying options for the $ENQ operation. The **flags** argument is a longword bit mask that is the logical OR of each bit set, where each bit corresponds to an option.

The $LCKDEF macro defines a symbolic name for each flag bit. The following table describes each flag.

Flag	Description
LCK$M_NOQUEUE	When this flag is specified, $ENQ does not queue the lock request unless the lock can be granted immediately. By default, $ENQ always queues the request. If you specify LCK$M_NOQUEUE in a lock conversion operation and the conversion cannot be granted immediately, the lock remains in the original lock mode.

Flag	Description
LCK$M_SYNCSTS	When you specify this flag, $ENQ returns the successful condition value SS$_SYNCH in R0 if the lock request is granted immediately; in this case, no completion AST is delivered and no event flag is set. If the lock request is queued successfully but cannot be granted immediately, $ENQ returns the condition value SS$_NORMAL in R0; then when the request is granted, $ENQ sets the event flag and queues an AST if the **astadr** argument was specified.
LCK$M_SYSTEM	When you specify this flag, the resource name is interpreted as systemwide. By default, resource names are qualified by the UIC group number of the creating process. This flag is ignored in lock conversions.
LCK$M_VALBLK	When you specify this flag, the lock status block contains a lock value block. See the description of the **lksb** argument for more information.
LCK$M_CONVERT	When you specify this flag, $ENQ performs a lock conversion. In this case, the caller must supply (in the second longword of the lock status block) the lock identification of the lock to be converted.
LCK$M_NODLCKWT	By specifying this flag, a process indicates to the lock management services that it is not blocked from execution while waiting for the lock request to complete. For example, a lock request might be left outstanding on the waiting queue as a signaling device between processes. This flag helps to prevent false deadlocks by providing the lock management services with additional information about the process issuing the lock request. When you set this flag, the lock management services do not consider this lock when trying to detect deadlock conditions. A process should specify the LCK$M_NODLCKWT flag only in a call to the $ENQ system service. The $ENQW system service waits for the lock request to be granted before returning to the caller; therefore, specifying the LCK$M_NODLCKWT flag in a call to the $ENQW system service defeats the purpose of the flag and can result in a genuine deadlock being ignored. The lock management services make use of the LCK$M_NODLCKWT flag only when the lock specified by the call to $ENQ is in either the waiting or the conversion queue. Improper use of the LCK$M_NODLCKWT flag can result in the lock management services ignoring genuine deadlocks.

Flag	Description
LCK$M_NODLCKBLK	By specifying this flag, a process indicates to the lock management services that, if this lock is blocking another lock request, the process intends to give up this lock on demand. When you specify this flag, the lock management services do not consider this lock as blocking other locks when trying to detect deadlock conditions. A process typically specifies the LCK$M_NODLCKBLK flag only when it also specifies a blocking AST. Blocking ASTs notify processes with granted locks that another process with an incompatible lock mode has been queued to access the same resource. Use of blocking ASTs may cause false deadlocks, because the lock management services detect a blocking condition, even though a blocking AST has been specified; however, the blocking condition will disappear as soon as the process holding the lock executes, receives the blocking AST, and dequeues the lock. Specifying the LCK$M_NODLCKBLK flag prevents this type of false deadlock. To enable blocking ASTs, the **blkast** argument of the $ENQ system service must contain the address of a blocking AST service routine. If the process specifies the LCK$M_NODLCKBLK flag, the blocking AST service routine should either dequeue the lock or convert it to a lower lock mode without issuing any new lock requests. If the blocking AST routine does otherwise, a genuine deadlock could be ignored. The lock management services make use of the LCK$M_NODLCKBLK flag only when the lock specified by the call to $ENQ has been granted. Improper use of the LCK$M_NODLCKBLK flag can result in the lock management services ignoring genuine deadlocks.
LCK$M_NOQUOTA	This flag is reserved by DIGITAL. When you set this flag, the calling process is not charged Enqueue Limit (ENQLM) quota for this new lock. The calling process must be running in executive or kernel mode to set this flag. This flag is ignored for lock conversions.
LCK$M_CVTSYS	This flag is reserved by DIGITAL. When you set this flag, the lock is converted from a process-owned lock to a system-owned lock. The calling process must be running in executive or kernel mode to set this flag.

resnam

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

Name of the resource to be locked by this lock. The **resnam** argument is the address of a character string descriptor pointing to this name. The name string may be from 1 to 31 bytes in length.

The **resnam** argument is required for new locks and is ignored for lock conversions.

parid

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

Lock identification of the parent lock. The **parid** argument is a longword containing this identification value.

If you do not specify this argument or specify it as 0, $ENQ assumes that the lock does not have a parent lock. This argument is optional for new locks and is ignored for lock conversions.

astadr

VMS usage: **ast_procedure**
type: **procedure entry mask**
access: **call without stack unwinding**
mechanism: **by reference**

AST service routine to be executed when the lock is either granted or converted. The **astadr** argument is the address of the entry mask of this routine.

If you specify the **astadr** argument, the AST routine executes at the same access mode as the caller of $ENQ.

astprm

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

AST parameter to be passed to the AST routine specified by the **astadr** argument. The **astprm** argument specifies this longword parameter.

blkast

VMS usage: **ast_procedure**
type: **procedure entry mask**
access: **call without stack unwinding**
mechanism: **by reference**

Blocking AST routine to be called whenever this lock is granted and is blocking any other lock requests. The **blkast** argument is the address of the entry mask to this routine.

You may pass a parameter to this routine by using the **astprm** argument.

acmode

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

Access mode to be associated with the resource name. For more information on the components of the resource name, see the Resource Names section in the "Lock Management Services" chapter of the *Introduction to VMS System*

Services. The **acmode** argument indicates the least privileged access mode from which locks can be queued on the resource.

This argument does not affect the access mode associated with the lock or its blocking and completion ASTs. 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$C_KERNEL	Kernel
PSL$C_EXEC	Executive
PSL$C_SUPER	Supervisor
PSL$C_USER	User

The $ENQ service associates an access mode with the lock in the following way:

- If you specified a parent lock (with the **parid** argument), $ENQ uses the access mode associated with the parent lock and ignores both the **acmode** argument and the caller's access mode.
- If the lock has no parent lock (you did not specify the **parid** argument or specified it as 0), $ENQ uses the least privileged of the caller's access mode and the access mode specified by the **acmode** argument. If you do not specify the **acmode** argument, $ENQ uses the caller's access mode.

nullarg

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

Place-holding argument reserved by DIGITAL.

DESCRIPTION

To queue a lock on a systemwide resource, the calling process must either have SYSLCK privilege or be executing in executive or kernel mode.

To specify a parent lock when queuing a lock, the access mode of the caller must be equal to, or less privileged than, the access mode associated with the parent lock.

To queue a lock conversion, the access mode associated with the lock being converted must be equal to, or less privileged than, the access mode of the calling process.

The $ENQ service uses the following system resources:

- Enqueue Limit (ENQLM) quota
- AST limit (ASTLM) quota in lock conversion requests that specify either the **astadr** or **blkast** argument
- System dynamic memory for the creation of the lock and resource blocks

When $ENQ queues a lock request, it returns the status of the request in R0 and writes the lock identification of the lock in the lock status block. Then, when the lock request is granted, $ENQ writes the final completion status in the lock status block, sets the event flag, and calls the AST routine, if this has been requested.

When $ENQW queues a lock request, it returns status in R0 and in the lock status block when the lock has been either granted or converted. At this time, it also sets the event flag and calls the AST routine, if this has been requested.

Invalidation of the Lock Value Block

In some situations, the lock value block may become invalid. In these situations, $ENQ warns the caller by returning the condition value SS$_VALNOTVALID in the lock status block, provided the caller has specified the flag LCK$M_VALBLK in the **flags** argument.

The SS$_VALNOTVALID condition value is a warning message, not an error message. Therefore, the $ENQ service will proceed to grant the requested lock. Further, $ENQ will return this warning on all subsequent calls to $ENQ until either a new lock value block is written to the lock database or the resource is deleted. Resource deletion occurs when no locks are associated with the resource.

The following events may cause the lock value block to become invalid:

- If any process holding a protected write or exclusive mode lock on a resource is terminated abnormally, the lock value block becomes invalid.
- If a VAX node in a VAXcluster fails and a process on that node was holding (or may have been holding) a protected write or exclusive mode lock on the resource, the lock value block becomes invalid.
- If a process holding a protected write or exclusive mode lock on the resource calls the Dequeue Lock Request ($DEQ) service to dequeue this lock and specifies the flag LCK$M_INVVALBLK in the **flags** argument, the lock value block maintained in the lock database is marked invalid.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully; the lock request was successfully queued.
SS$_SYNCH	The service completed successfully; the LCK$M_SYNCSTS flag in the **flags** argument was specified, and $ENQ was able to grant the lock request immediately.
SS$_ACCVIO	The lock status block or the resource name cannot be read.
SS$_BADPARAM	You specified an invalid lock mode in the **lkmode** argument.
SS$_CVTUNGRANT	You attempted a lock conversion on a lock that is not currently granted.
SS$_EXDEPTH	The limit of levels of sublocks has been exceeded.
SS$_EXENQLM	The process has exceeded its Enqueue Limit (ENQLM) quota.

SS$_INSFMEM	The system dynamic memory is insufficient for creating the necessary data structures.
SS$_IVBUFLEN	The length of the resource name was either 0 or greater than 31.
SS$_IVLOCKID	You specified an invalid or nonexistent lock identification, or the lock identified by the lock identification has an associated access mode that is more privileged than the caller's, or the access mode of the parent was less privileged than that of the caller.
SS$_NOLOCKID	No lock identification was available for the lock request.
SS$_NOTQUEUED	The lock request was not queued; the LCK$M_NOQUEUE flag in the **flags** argument was specified and $ENQ was not able to grant the lock request immediately.
SS$_NOSYSLCK	The LCK$M_SYSTEM flag in the **flags** argument was specified but the caller lacks the necessary SYSLCK privilege.
SS$_PARNOTGRANT	The parent lock specified in the **parid** argument was not granted.

CONDITION VALUES RETURNED IN THE LOCK STATUS BLOCK

SS$_NORMAL	The service completed successfully; the lock was successfully granted or converted.
SS$_ABORT	The lock was dequeued (by the $DEQ service) before $ENQ could grant the lock.
SS$_DEADLOCK	A deadlock was detected.
SS$_CANCEL	The lock conversion request has been canceled and the lock has been regranted at its previous lock mode. This condition value is returned when $ENQ queues a lock conversion request, the request has not been granted yet (it is in the conversion queue), and, in the interim, the $DEQ service is called (with the LCK$M_CANCEL flag specified) to cancel this lock conversion request. If the lock is granted before $DEQ can cancel the conversion request, the call to $DEQ returns the condition value SS$_CANCELGRANT, and the call to $ENQ returns SS$_NORMAL.
SS$_VALNOTVALID	The lock value block is marked invalid. This warning message is returned only if the caller has specified the flag LCK$M_VALBLK in the **flags** argument. Note that the lock has been successfully granted despite the return of this warning message. Refer to the Description section for a complete discussion of lock value block invalidation.

$ENQW Enqueue Lock Request and Wait

The Enqueue Lock Request and Wait service queues a lock on a resource. The $ENQW service completes synchronously; that is, it returns to the caller when the lock has been either granted or converted. For asynchronous completion, use the Enqueue Lock Request ($ENQ) service; $ENQ returns to the caller after queueing the lock request, without waiting for the lock to be either granted or converted. In all other respects, $ENQW is identical to $ENQ. Refer to the documentation of $ENQ for all other information about the $ENQW service.

For additional information about system service completion, refer to the documentation of the Synchronize ($SYNCH) service and to the *Introduction to VMS System Services*.

The $ENQ, $ENQW, $DEQ, and $GETLKI services together provide the user interface to the VMS lock management facility. For additional information about lock management, refer to the descriptions of these other services and to the *Introduction to VMS System Services*.

FORMAT

SYS$ENQW *[efn] ,lkmode ,lksb [,flags] [,resnam] [,parid] [,astadr] [,astprm] [,blkast] [,acmode] [,nullarg]*

$ERAPAT Get Security Erase Pattern

The Get Security Erase Pattern service generates a security erase pattern. A user-written erase routine can then write this pattern into areas of memory containing sensitive data that is no longer in use to prevent the inadvertent disclosure of the sensitive data.

FORMAT

SYS$ERAPAT *[type],[count],[patadr]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

type

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

Type of storage to be written over with the erase pattern. The **type** argument is a longword containing the type of storage. The three storage types and their symbolic names (defined by the $ERADEF macro) follow:

Storage Type	Symbolic Name
Main memory	ERA$K_MEMORY
Disk	ERA$K_DISK
Tape	ERA$K_TAPE

count

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

The number of times that $ERAPAT has been called in a single security erase operation. The **count** argument is a longword containing the iteration count.

You should call the $ERAPAT service initially with the **count** argument set to 1, the second time with the **count** argument set to 2, and so on, until the status code SS$_NOTRAN is returned.

patadr

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

The security erase pattern to be written. The **patadr** argument is the address of a longword into which the security erase pattern is to be written.

DESCRIPTION

The $ERAPAT service provides a consistent mechanism for performing security erase operations. This service is used primarily by VMS, but it may also be used by users who want to perform security erase operations on foreign disks.

You should call the $ERAPAT service iteratively until the completion status SS$_NOTRAN is returned.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully; proceed with the next erase step.
SS$_NOTRAN	The service completed successfully; security erase completed.
SS$_BADPARAM	The **type** argument or **count** argument is invalid.
SS$_ACCVIO	The **patadr** argument cannot be written by the caller.

EXAMPLE

The following example demonstrates how to use the $ERAPAT service to perform a security erase to a disk. Note that after each call to $ERAPAT, a test for the status SS$_NOTRAN is made. If SS$_NOTRAN has not been returned, $QIO is called to write the pattern returned by $ERAPAT onto the disk. After this write, $ERAPAT is called again and the cycle is repeated until the code SS$_NOTRAN is returned, at which point the security erase procedure is complete.

```
; Code fragment that erases 20 blocks (blocks 15 through 34)
; on a disk
;
PATTERN:
        .LONG   0                       ; Cell to contain output from $ERAPAT
CHANNEL:
        .WORD   0                       ; Channel assigned to disk device
DEVICE: .ASCID  /DISK:/                 ; Disk device name
                .
                .
                .
        $ASSIGN_S DEVNAM=DISK,-         ; Assign a channel to the device
                CHAN=CHANNEL
        BLBC    R0, EXIT                ; Branch if error
                .
                .
                .
        MOVL    #1, R2                  ; Set initial count
        $ERADEF                         ; Macro to define names
                                        ; used by $ERAPAT
```

```
10$:    $ERAPAT_S -                     ; Call the $ERAPAT service
                COUNT=R2,-
                TYPE=#ERA$K_DISK,-
                PATADR=PATTERN
        BLBC    R0, EXIT                ; Branch if error
        CMPL    #SS$_NOTRAN, R0         ; Are we done?
        BEQL    EXIT                    ; Branch if so
        $QIO_S  CHAN=CHANNEL,-
                FUNC=#IO$_WRITELBLK!IO$M_ERASE,-        ; Call
                P1=PATTERN,-            ; to the $QIO service
                P2=#<20*512>,-          ; to write the erase
                P3=#15                  ; pattern

        INCL    R2                      ; Increase count

        BRB     10$

EXIT:           .
                .
                .
```

$EXIT Exit

The Exit service is used by the operating system to initiate image rundown when the current image in a process completes execution. Control normally returns to the command interpreter.

FORMAT

SYS$EXIT *[code]*

ARGUMENT

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

Longword value to be saved in the process header as the completion status of the current image. If you do not specify this argument in a macro call, a value of 1 is passed as the completion code for VAX MACRO and VAX BLISS-32, and a value of 0 is passed for other languages. You can test this value at the command level to provide conditional command execution.

DESCRIPTION

The $EXIT service is unlike all other system services in that it does not return status codes in R0 or anywhere else. The $EXIT service does not return control to the caller; it performs an exit to the command interpreter or causes the process to terminate if no command interpreter is present.

For a summary of the actions taken by the system at image exit, see the *Introduction to VMS System Services.*

CONDITION VALUES RETURNED

None

$EXPREG Expand Program/Control Region

The Expand Program/Control Region service adds a specified number of new virtual pages to a process's program region or control region for the execution of the current image. Expansion occurs at the current end of that region's virtual address space.

FORMAT

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

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns 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 add to the current end of the program or control region. The **pagcnt** argument is a longword value containing this number.

retadr

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

Starting and ending process virtual addresses of the pages that $EXPREG has actually added. The **retadr** argument is the address of a 2-longword array containing, in order, the starting and ending process virtual addresses.

acmode

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

Access mode to be associated with the newly added pages. The **acmode** argument is a longword containing the access mode.

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

The newly added pages are given the following protection: (1) read and write access for access modes equal to or more privileged than the access mode used in the call and (2) no access for access modes less privileged than that used in the call.

region

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

Number specifying which program region is to be expanded. The **region** argument is a longword value. A value of 0 (the default) specifies that the program region (P0 region) is to be expanded. A value of 1 specifies that the control region (P1 region) is to be expanded.

DESCRIPTION

The process's paging file quota (PGFLQUOTA) must be sufficient to accommodate the increased size of the virtual address space.

The new pages, which were previously inaccessible to the process, are created as demand-zero pages.

Because the bottom of the user stack is normally located at the end of the control region, expanding the control region is equivalent to expanding the user stack. The effect is to increase the available stack space by the specified number of pages.

The starting address returned is always the first available page in the designated region; therefore, the ending address is smaller than the starting address when the control region is expanded and is larger than the starting address when the program region is expanded.

If an error occurs while pages are being added, the **retadr** argument (if specified) indicates the pages that were successfully added before the error occurred. If no pages were added, both longwords of the **retadr** argument contain the value –1.

The information returned in the location addressed by the **retadr** argument (if specified) can be used as the input range to the Delete Virtual Address Space ($DELTVA) service.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The return address array cannot be written by the caller.
SS$_EXQUOTA	The process exceeded its paging file quota.
SS$_ILLPAGCNT	The specified page count was less than 1.
SS$_INSFWSL	The process's working set limit is not large enough to accommodate the increased virtual address space.
SS$_VASFULL	The process's virtual address space is full. No space is available in the process page table for the requested regions.

$FAO Formatted ASCII Output

The Formatted ASCII Output service (1) converts a binary value into an ASCII character string in decimal, hexadecimal, or octal notation and returns the character string in an output string and (2) inserts variable character string data into an output string.

The Formatted ASCII Output with List Parameter ($FAOL) service provides an alternate way to specify input parameters for a call to the $FAO system service. The formats for both $FAO and $FAOL are shown under FORMAT.

FORMAT

SYS$FAO *ctrstr ,[outlen] ,outbuf ,[p1]...[pn]*
SYS$FAOL *ctrstr ,[outlen] ,outbuf [,prmlst]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

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

Control string passed to $FAO that contains the text to be output together with one or more FAO directives. The **ctrstr** argument is the address of a character string descriptor pointing to the control string. The FAO directives are described in the Description section.

There is no restriction on the length of the control string, nor on the number of FAO directives it may contain. However, if an exclamation point (!) must appear in the output string, it must be represented in the control string by a double exclamation point (!!). A single exclamation point in the control string indicates to $FAO that the next characters are to be interpreted as FAO directives.

When $FAO processes the control string, it writes each character that is not part of an FAO directive to the output buffer.

If the FAO directive is valid, $FAO processes it. If the directive requires a parameter, $FAO processes the next consecutive parameter in the specified parameter list. If the FAO directive is not valid, $FAO terminates and returns a condition value in R0.

The $FAO service reads parameters from the argument list specified in the call; these arguments have the names **p1**, **p2**, **p3**, and so on, up to **p20**. Each argument specifies one parameter. Because $FAO accepts a maximum of 20 parameters in a single call, you must use the $FAOL service if the number of parameters exceeds 20. The $FAOL service accepts any number of parameters used with the **prmlst** argument.

outlen

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

Length in bytes of the fully formatted output string returned by $FAO. The **outlen** argument is the address of a word containing this value.

outbuf

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

Output buffer into which $FAO writes the fully formatted output string. The **outbuf** argument is the address of a character string descriptor pointing to the output buffer.

p1 to pn

VMS usage: **varying_arg**
type: **longword (signed)**
access: **read only**
mechanism: **by value**

FAO directive parameter(s). The **p1** argument is a longword containing the parameter needed by the first FAO directive encountered in the control string, the **p2** argument is a longword containing the parameter needed for the second FAO directive, and so on for the remaining arguments up to **p20**. If an FAO directive does not require a parameter, that FAO directive is processed without reading a parameter from the argument list.

Depending on the directive, a parameter may be a value to be converted, an address of a string to be inserted into the output string, or a length or argument count. Each directive in the control string may require a corresponding parameter or parameters.

prmlst

VMS usage: **vector_longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

List of FAO directive parameters to be passed to the $FAOL service. The **prmlst** argument is the address of a list of longwords wherein each longword is a parameter. The $FAOL service processes these parameters sequentially as it encounters, in the control string, FAO directives that require parameters.

The parameter list may be a data structure that already exists in a program and from which certain values are to be extracted.

DESCRIPTION

The $FAO_S macro form uses a PUSHL instruction for all parameters (**p1** through **pn**) passed to the service; if you specify a symbolic address, it must be preceded by a number sign character (#) or loaded into a register.

You can specify a maximum of 20 parameters on the $FAO macro. If more than 20 parameters are required, use the $FAOL macro.

This service does not check the length of the argument list, and therefore cannot return the SS$_INSFARG (insufficient arguments) error status code. If the service does not receive enough arguments (for example, if you omit required commas in the call), you may not get the desired result.

Format of FAO Directives

FAO directives may appear anywhere in the control string. The general format of an FAO directive is as follows:

```
!DD
```

The exclamation point (!) specifies that the following character(s) are to be interpreted as an FAO directive and the characters "DD" represent a 1- or 2-character FAO directive. When the characters of the FAO directive are alphabetic, they must be uppercase.

An FAO directive may optionally specify the following:

- A repeat count. The format is as follows:

  ```
  !n(DD)
  ```

  In this case *n* is a decimal value specifying the number of times that $FAO is to repeat the directive. If the directive requires a parameter or parameters, $FAO uses successive parameters from the parameter list for each repetition of the directive; it does not use the same parameter(s) for each repetition. The parentheses are required syntax.

- An output field length. The format is as follows:

  ```
  !mDD
  ```

  In this case *m* is a decimal value specifying the length of the field (within the output string) into which $FAO is to write the output resulting from the directive. The length is expressed as a number of characters.

- Both a repeat count and output field length. In this case the format is as follows:

  ```
  !n(mDD)
  ```

Specifying Variables for Repeat Count and Field Length

You may specify repeat counts and output field lengths as variables by using a number sign (#) in place of an absolute numeric value. If you specify a number sign (#) for a repeat count, the next parameter passed to FAO must contain the count. If you specify a number sign (#) for an output field length, the next parameter must contain the length value.

If you specify a number sign (#) for both the output field length and for the repeat count, only one length parameter is required; each output string will have the specified length.

If you specify a number sign (#) for the repeat count, the output field length, or both, the parameter(s) specifying the count, length, or both, must precede other parameters required by the directive.

Table SYS–3 lists and describes the FAO directives.

Table SYS–3 List of FAO Directives

Directive	Description
Directives for Character String Substitution	
!AC	Inserts a counted ASCII string. It requires one parameter: the address of the string to be inserted. The first byte of the string must contain the length in characters of the string.
!AD	Inserts an ASCII string. It requires two parameters: the length of the string and the address of the string. Each of these parameters is a separate argument.
!AF	Inserts an ASCII string and replaces all nonprintable ASCII codes with periods (.). It requires two parameters: the length of the string and the address of the string. Each of these parameters is a separate argument.
!AS	Inserts an ASCID string. It requires one parameter: the address of a character string descriptor pointing to the string.
Directives for Zero-Filled Numeric Conversion	
!OB	Converts a byte value to the ASCII representation of the value's octal equivalent. It requires one parameter: the value to be converted. $FAO uses only the low-order byte of the longword parameter.
!OW	Converts a word value to the ASCII representation of the value's octal equivalent. It requires one parameter: the value to be converted. $FAO uses only the low-order word of the longword parameter.
!OL	Converts a longword value to the ASCII representation of the value's octal equivalent. It requires one parameter: the value to be converted.
!XB	Converts a byte value to the ASCII representation of the value's hexadecimal equivalent. It requires one parameter: the value to be converted. $FAO uses only the low-order byte of the longword parameter.
!XW	Converts a word value to the ASCII representation of the value's hexadecimal equivalent. It requires one parameter: the value to be converted. $FAO uses only the low-order word of the longword parameter.
!XL	Converts a longword value to the ASCII representation of the value's hexadecimal equivalent. It requires one parameter: the value to be converted.

Table SYS–3 (Cont.) List of FAO Directives

Directive	Description
Directives for Zero-Filled Numeric Conversion	
!ZB	Converts an unsigned byte value to the ASCII representation of the value's decimal equivalent. It requires one parameter: the value to be converted. $FAO uses only the low-order byte of the longword parameter.
!ZW	Converts an unsigned word value to the ASCII representation of the value's decimal equivalent. It requires one parameter: the value to be converted. $FAO uses only the low-order word of the longword parameter.
!ZL	Converts an unsigned longword value to the ASCII representation of the value's decimal equivalent. It requires one parameter: the value to be converted.
Directives for Blank-Filled Numeric Conversion	
!UB	Converts an unsigned byte value to the ASCII representation of the value's decimal equivalent. It requires one parameter: the value to be converted. $FAO uses only the low-order byte of the longword parameter.
!UW	Converts an unsigned word value to the ASCII representation of the value's decimal equivalent. It requires one parameter: the value to be converted. $FAO uses only the low-order word of the longword parameter.
!UL	Converts an unsigned longword value to the ASCII representation of the value's decimal equivalent. It requires one parameter: the value to be converted.
!SB	Converts a signed byte value to the ASCII representation of the value's decimal equivalent. It requires one parameter: the value to be converted. $FAO uses only the low-order byte of the longword parameter.
!SW	Converts a signed word value to the ASCII representation of the value's decimal equivalent. It requires one parameter: the value to be converted. $FAO uses only the low-order word of the longword parameter.
!SL	Converts a signed longword value to the ASCII representation of the value's decimal equivalent. It requires one parameter: the value to be converted.
Directives for Output String Formatting	
!/	Inserts a new line, that is, a carriage return and line feed. It takes no parameters.
!_	Inserts a tab. It takes no parameters.
!^	Inserts a form feed. It takes no parameters.
!!	Inserts an exclamation point. It takes no parameters.

Table SYS–3 (Cont.) List of FAO Directives

Directive	Description
Directives for Output String Formatting	
!%S	Inserts the letter "S" if the most recently converted numeric value is not 1. An uppercase "S" is inserted if the character before the !%S directive is an uppercase character; a lowercase "s" is inserted if the character is lowercase.
!%T	Inserts the system time. It takes one parameter: the address of a quadword time value to be converted to ASCII. If you specify 0, the current system time is inserted.
!%U	Converts a longword integer UIC to a standard UIC specification in the format [xxx,yyy], where *xxx* is the group number and *yyy* is the member number. It takes one parameter: a longword integer. The directive inserts the surrounding brackets ([ ]) and comma (,).
!%I	Converts a longword to the appropriate alphanumeric identifier. If the longword represents a UIC, surrounding brackets ([ ]) and comma (,) are added as necessary. If no identifier exists and the longword represents a UIC, the longword is formatted as in !%U. Otherwise it is formatted as in !XL with a preceding !%X added to the formatted result.
!%D	Inserts the system date and time. It takes one parameter: the address of a quadword time value to be converted to ASCII. If you specify 0, the current system date and time are inserted.
!n <	See description of next directive (!>).
!>	This directive and the preceding one (!n <) are used together to define an output field width of *n* characters within which all data and directives to the right of !n < and to the left of !> are left-justified and blank-filled. It takes no parameters.
!n*c	Repeats the character *c* in the output string *n* times.
Directives for Parameter Interpretation	
!-	Causes $FAO to reuse the most recently used parameter in the list. It takes no parameters.
!+	Causes $FAO to skip the next parameter in the list. It takes no parameters.

Table SYS–4 shows the FAO output field lengths and their fill characters.

Table SYS–4 FAO Output Field Lengths and Fill Characters

Conversion/Substitution Type	Default Length of Output Field	Action When Explicit Output Field Length Is Longer than Default	Action When Explicit Output Field Length Is Shorter than Default
Hexadecimal Byte Word Longword	 2 (zero-filled) 4 (zero-filled) 8 (zero-filled)	ASCII result is right-justified and blank-filled to the specified length	ASCII result is truncated on the left
Octal Byte Word Longword	 3 (zero-filled) 6 (zero-filled) 11 (zero-filled)	Hexadecimal or octal output is always zero-filled to the default output field length then blank-filled to specified length	
Signed or Unsigned Decimal	As many characters as necessary	ASCII result is right-justified and blank-filled to the specified length	Signed and unsigned decimal output fields and completely filled with asterisks (*)
Unsigned Zero-filled Decimal	As many characters as necessary	ASCII result is right-justified and zero-filled to the specified length	
ASCII String Substitution	Length of input character string	ASCII string is left-justified and blank-filled to the specified length	ASCII string is truncated on the right

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_BUFFEROVF	The service completed successfully. The formatted output string overflowed the output buffer and has been truncated.
SS$_ACCVIO	The **ctrstr**, **p1** through **pn**, or **prmlst** argument cannot be read, or the **outlen** argument cannot be written (it may specify 0).
SS$_BADPARAM	You specified an invalid directive in the FAO control string.

EXAMPLES

Each of the following examples shows an FAO control string with several directives, parameters defined as input for the directives, and the calls to $FAO to format the output strings. The numbered examples illustrate the following:

1 $FAO macro, !AC, !AS, !AD, and !/ directives

2 $FAO macro, !!, and !AS directives, repeat count, output field length

3 $FAO macro, !UL, !XL, !SL directives

4 $FAOL macro, !UL, !XL, !SL directives

5 $FAOL macro, !UB, !XB, !SB directives

6 $FAO macro, !XW, !ZW, !- directives, repeat count, output field length

7 $FAOL macro, !AS, !UB, !%S, !- directives, variable repeat count

8 $FAO macro, !nc(repeat character), !%D directives

9 $FAO macro, !%D and !%T (with output field lengths), !n (with variable repeat count)

10 $FAO macro, ! < and !> (define field width), !AC, and !UL directives

11 $FAO macro, !AS and !SL directives

Each example is accompanied by notes. These notes show the output string created by the call to $FAO and describe in more detail some considerations for using directives. The sample output strings show a delta character (_) for each space in all places where FAO output contains multiple spaces.

Each of the first 10 examples refers to the following output fields (these fields are not shown in the data areas for each example):

```
FAODESC:                    ; Descriptor for output buffer
        .LONG   80                  ; Output buffer length
        .ADDRESS -
                FAOBUF              ; Address of buffer
FAOBUF: .BLKB   80                  ; 80-character buffer
FAOLEN: .BLKW   1                   ; Receive length of output
        .BLKW   1                   ; Reserve word for $QIO
```

These examples assume that each call to $FAO will be followed by a call to $QIO to write the output string produced by $FAO. The $QIO system service requires that the length be specified as a longword; therefore, an extra word is provided following the word defined to receive the length of the output string returned by $FAO.

The final example shows how to make a call to $FAO from a VAX FORTRAN program.

The examples, numbered 1 through 11, follow.

1

```
    ; Control String and input parameters
    ;
    SLEEPSTR: .ASCID "!/SAILORS: !AC !AS !AD" ; Descriptor for control
                                              ; string
    ;
    WINKEN: .ASCIC  /WINKEN/                  ; Counted ASCII string
    BLINKEN:
            .ASCID  /BLINKEN/                 ; Character string descriptor
    NOD:    .ASCII  /NOD/                     ; ASCII string
    NODLEN: .LONG   NODLEN-NOD                ; Length of ASCII string
              .
              .
              .
    ; Call to $FAO
    ;
            $FAO_S  CTRSTR=SLEEPSTR, -
                    OUTLEN=FAOLEN, -
                    OUTBUF=FAODESC,-
                    P1=#WINKEN, -
                    P2=#BLINKEN, -
                    P3=NODLEN, -
                    P4=#NOD
```

$FAO writes the following output string into FAOBUF:

```
<CR><KEY>(LF\TEXT)SAILORS: WINKEN BLINKEN NOD
```

The !/ directive provides a carriage-return/line-feed character (shown as <CR> <KEY> (LF\TEXT)) for terminal output.

The !AC directive requires the address of a counted ASCII string (**p1** argument); the number sign (#) is required to specify the parameter, so that the PUSHL instruction used by the $FAO macro pushes the address rather than its contents.

The !AS directive requires the address of a character string descriptor (**p2** argument).

The !AD directive requires two parameters: the length of the string to be substituted (**p3** argument) and its address (**p4** argument).

2
```
; Control string and input parameters
;
NAMESTR:
        .ASCID  /UNABLE TO LOCATE !3(8AS)!!/ ; Descriptor for
                                             ; control string
;
JONES:  .ASCID  /JONES/                 ; Name descriptor
HARRIS: .ASCID  /HARRIS/                ; Name descriptor
WILSON: .ASCID  /WILSON/                ; Name descriptor
          .
          .
          .
; Call to $FAO
;
        $FAO_S  CTRSTR=NAMESTR, -
                OUTLEN=FAOLEN, -
                OUTBUF=FAODESC,-
                P1=#JONES, -
                P2=#HARRIS, -
                P3=#WILSON
```

$FAO writes the following output string into FAOBUF:

```
UNABLE TO LOCATE JONES___HARRIS__WILSON__!
```

The !3(8AS) directive contains a repeat count: three parameters (addresses of character string descriptors) are required. $FAO left-justifies each string into a field of eight characters (the output field length specified).

The double exclamation point directive (!!) supplies a literal exclamation point (!) in the output.

If the directive were specified without an output field length, that is, if the directive were specified as !3(AS), the three output fields would be concatenated, as follows:

```
UNABLE TO LOCATE JONESHARRISWILSON!
```

3
```
; Control strings and input parameters for next three examples
;
; Descriptor for control string (longword conversion)
LONGSTR:
        .ASCID  /VALUES !UL (DEC) !XL (HEX) !SL (SIGNED)/
;
; Descriptor for control string (byte conversion)
BYTESTR:
        .ASCID  /VALUES !UB (DEC) !XB (HEX) !SB (SIGNED)/
;
VAL1:   .LONG   200                     ; Decimal 200
VAL2:   .LONG   300                     ; Decimal 300
VAL3:   .LONG   -400                    ; Negative 400
        .
        .
        .
;
; Example 3:  Call to $FAO
;
        $FAO_S  CTRSTR=LONGSTR, -
                OUTBUF=FAODESC, -
                OUTLEN=FAOLEN,-
                P1=VAL1, -
                P2=VAL2, -
                P3=VAL3
```

$FAO writes the following output string:

```
VALUES 200 (DEC) 0000012C (HEX) -400 (SIGNED)
```

The longword value 200 is converted to decimal, the value 300 is converted to hexadecimal, and the value –400 is converted to signed decimal. The ASCII results of each conversion are placed in the appropriate position in the output string.

Note that the hexadecimal output string has eight characters and is zero-filled to the left. This is the default output length for hexadecimal longwords.

4
```
;
;       Call to $FAOL
;
        $FAOL_S CTRSTR=LONGSTR, -
                OUTBUF=FAODESC, -
                OUTLEN=FAOLEN,-
                PRMLST=VAL1
```

$FAO writes the following output string:

```
VALUES 200 (DEC) 0000012C (HEX) -400 (SIGNED)
```

The results are the same as the results of Example 3. However, unlike the $FAO macro, which requires each parameter on the call to be specified, the $FAOL macro points to a list of consecutive longwords, which $FAO reads as parameters.

```
5  ;
   ;       Call to $FAOL
   ;
           $FAOL_S CTRSTR=BYTESTR, -
                   OUTLEN=FAOLEN, -
                   OUTBUF=FAODESC,-
                   PRMLST=VAL1
```

$FAO writes the following output string:

```
VALUES 200 (DEC) 2C (HEX) 112 (SIGNED)
```

The input parameters are the same as those for Example 4. However, the control string (BYTESTR) specifies that byte values are to be converted. $FAO uses the low-order byte of each longword parameter passed to it. The high-order three bytes are not evaluated. Compare these results with the results of Example 4.

```
6  ;
   ; Control string
   ;
   MULTSTR:
           .ASCID /HEX: !2(6XW) ZERO-DEC: !2(-)!2(7ZW)/
             .
             .
             .
   ; Call to $FAO
   ;
           $FAO_S  CTRSTR=MULTSTR, -
                   OUTLEN=FAOLEN, -
                   OUTBUF=FAODESC,-
                   P1=#10000, -
                   P2=#9999
```

FAO writes the following output string:

```
HEX:___2710__270F ZERO-DEC: 00100000009999
```

Each of the directives !2(6XW) and !2(7ZW) contains repeat counts and output lengths. First, $FAO performs the !XW directive twice, using the low-order word of the numeric parameters passed. The output length specified is two characters longer than the default output field width of hexadecimal word conversion, so two spaces are placed between the resulting ASCII strings.

The !- directive causes $FAO to back up over the parameter list. A repeat count is specified with the directive so that $FAO skips back over two parameters; then, it uses the same two parameters for the !ZW directive. The !ZW directive causes the output string to be zero-filled to the specified length, in this example, of seven characters. Thus, there are no spaces between the output fields.

```
;
; Control string and input parameters
;
ARGSTR: .ASCID /!AS RECEIVED !UB ARG!%S: !-!#(4UB)/
;
LISTA:  .ADDRESS -
                ORION                   ; Address of name string
        .LONG   3                       ; Number of args in list
        .LONG   10                      ; Argument 1
        .LONG   123                     ; Argument 2
        .LONG   210                     ; Argument 3
;
LISTB:  .ADDRESS -
                LYRA                    ; Address of name string
        .LONG   1                       ; Number of args in list
        .LONG   255                     ; Argument 1
;
ORION:  .ASCID  /ORION/                 ; Descriptor for process ORION
;
LYRA:   .ASCID  /LYRA/                  ; Descriptor for process LYRA
          .
          .
          .
; Calls to $FAO
;
        $FAOL_S CTRSTR=ARGSTR, -
                OUTLEN=FAOLEN, -
                OUTBUF=FAODESC,-
                PRMLST=LISTA
;
        $FAOL_S CTRSTR=ARGSTR, -
                OUTLEN=FAOLEN, -
                OUTBUF=FAODESC,-
                PRMLST=LISTB
```

After the first call to $FAOL, $FAO writes the following output strin:

```
ORION RECEIVED 3 ARGS:___10 123 210
```

Following the second call, $FAO writes the following output string:

```
LYRA RECEIVED 1 ARG:__255
```

In each of the examples, the PRMLST argument points to a different parameter list; each list contains, in the first longword, the address of a character string descriptor. The second longword begins an argument list, with the number of arguments remaining in the list. The control string uses this second longword twice: first to output the value contained in the longword, and then to provide the repeat count to output the number of arguments in the list (the !- directive indicates that $FAO should reuse the parameter).

The !%S directive provides a conditional plural. When the last value converted has a value not equal to 1, $FAO outputs the character "S"; if the value is a 1 (as in Example 2), $FAO does not output the character "S".

The output field length defines a width of four characters for each byte value converted, to provide spacing between the output fields.

8
```
;
; Control string
;
TIMESTR:
        .ASCID  /!5> NOW IS: !%D/
          .
          .
          .
; Call to $FAO
;
        $FAO_S CTRSTR=TIMESTR, -
               OUTLEN=FAOLEN, -
               OUTBUF=FAODESC,-
               P1=#0
```

FAO writes the following output string:

```
>>>>> NOW IS: dd-mmm-yyyy hh:mm:ss.cc
```

where:

dd	Is the day of the month.
mmm	Is the month.
yyyy	Is the year.
hh:mm:ss.cc	Is the time in hours, minutes, seconds, and hundredths of seconds.

The !5> directive requests $FAO to write five greater-than (>) characters into the output string. Because there is a space after the directive, $FAO also writes a space after the greater-than (>) characters on output.

The !%D directive requires the address of a quadword time value, which must be in the system time format. However, when the address of the time value is specified as 0, $FAO uses the current date and time. For information on how to obtain system time values in the required format, see the *Introduction to VMS System Services*. For a detailed description of the ASCII date and time string returned, see the discussion of the Convert Binary Time to ASCII String ($ASCTIM) system service.

9
```
;
; Control string
;
DAYTIMSTR:
         .ASCID  /DATE: !11%D!#_TIME: !5%T/
           .
           .
           .
; Call to $FAO
;
         $FAO_S CTRSTR=DAYTIMSTR, -
                OUTLEN=FAOLEN, -
                OUTBUF=FAODESC,-
                P1=#0, -
                P2=#5, -
                P3=#0
```

FAO writes the following output string:

```
DATE: dd-mmm-yyyy_____TIME: hh:mm
```

An output length of eleven bytes is specified with the !%D directive so that $FAO truncates the time from the date and time string, and outputs only the date.

The !#_ directive requests that the underscore character (_) be repeated the number of times specified by the next parameter. Because **p2** is specified as 5, five underscores are written into the output string.

The !%T directive normally returns the full system time. The !5%T directive provides an output length for the time; only the hours and minutes fields of the time string are written into the output buffer.

10
```
;
; Control string and parameters
;
WIDTHSTR:
        .ASCID  /!25<VAR: !AC VAL: !UL!>TOTAL:!7UL/
;
VAR1NAME:
        .ASCIC  /INVENTORY/             ; Variable 1 name
VAR1:   .LONG   334                     ; Current value
VAR1TOT:
        .LONG   6554                    ; Var 1 total
;
VAR2NAME:
        .ASCIC  /SALES/                 ; Var 2 name
VAR2:   .LONG   280                     ; Current value
VAR2TOT:
        .LONG   10750                   ; Var 2 total
          .
          .
          .
;
; Calls to $FAO
;
        $FAO_S  CTRSTR=WIDTHSTR, -
                OUTLEN=FAOLEN, -
                OUTBUF=FAODESC,-
                P1=#VAR1NAME, -
                P2=VAR1, -
                P3=VAR1TOT
;
        $FAO_S  CTRSTR=WIDTHSTR, -
                OUTLEN=FAOLEN, -
                OUTBUF=FAODESC,-
                P1=#VAR2NAME, -
                P2=VAR2, -
                P3=VAR2TOT
```

After the first call to $FAO, $FAO writes the following output string:

```
VAR: INVENTORY VAL: 334__TOTAL:___6554
```

After the second call, $FAO writes the following output string:

```
VAR: SALES VAL: 280______TOTAL:__10750
```

The !25 < directive requests an output field width of 25 characters; the end of the field is delimited by the !> directive. Within the field defined are two directives, !AC and !UL. The strings substituted by these directives can vary in length, but the entire field always has 25 characters.

The !7UL directive formats the longword passed in each example (**p2** argument) and right-justifies the result in a 7-character output field.

11

```
INTEGER STATUS,
2       SYS$FAO,
2       SYS$FAOL

! Resultant string
CHARACTER*80 OUTSTRING
INTEGER*2    LEN
! Array for directives in $FAOL
INTEGER*4    PARAMS(2)

! File name and error number
CHARACTER*80 FILE
INTEGER*4    FILE_LEN,
2            ERROR
! Descriptor for $FAOL
INTEGER*4    DESCR(2)

! These variables would generally be set following an error
FILE = '[BOELITZ]TESTING.DAT'
FILE_LEN = 18
ERROR = 25

! Call $FAO
STATUS = SYS$FAO ('File !AS aborted at error !SL',
2                 LEN,
2                 OUTSTRING,
2                 FILE(1:FILE_LEN),
2                 %VAL(ERROR))
IF (.NOT. STATUS) CALL LIB$SIGNAL (%VAL(STATUS))

TYPE *,'From SYS$FAO:'
TYPE *,OUTSTRING (1:LEN)

! Set up descriptor for filename
DESCR(1) = FILE_LEN    ! Length
DESCR(2) = %LOC(FILE)  ! Address
! Set up array for directives
PARAMS(1) = %LOC(DESCR) ! File name
PARAMS(2) = ERROR       ! Error number
! Call $FAOL
STATUS = SYS$FAOL ('File !AS aborted at error !SL',
2                  LEN,
2                  OUTSTRING,
2                  PARAMS)
IF (.NOT. STATUS) CALL LIB$SIGNAL (%VAL(STATUS))

TYPE *,'From SYS$FAOL:'
TYPE *,OUTSTRING (1:LEN)

END
```

This example shows a segment of a VAX FORTRAN program used to output the following string:

```
FILE [BOELITZ]TESTING.DAT ABORTED AT ERROR 25
```

$FILESCAN Scan String for File Specification

The Scan String for File Specification service searches a string for a file specification and parses the components of that file specification.

FORMAT

SYS$FILESCAN *srcstr ,valuelst ,[fldflags]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

srcstr

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

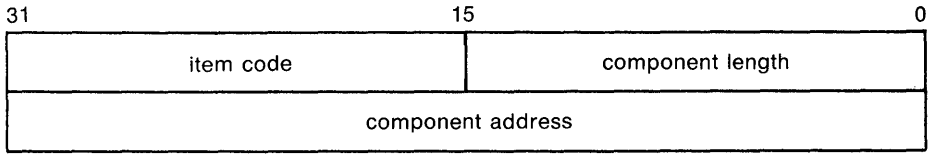
String to be searched for the file specification. The **srcstr** argument is the address of a descriptor pointing to this string.

valuelst

VMS usage: **item_list_2**
type: **longword (unsigned)**
access: **modify**
mechanism: **by reference**

Item list specifying which components of the file specification are to be returned by $FILESCAN. The components are the node, device, directory, file name, file type, and version number. The **itmlst** argument is the address of a list of item descriptors wherein each item descriptor specifies one component. The list of item descriptors is terminated by a longword of 0.

The following diagram depicts a single item descriptor.



ZK-1709-84

$FILESCAN Item Descriptor Fields

component length
A word in which $FILESCAN writes the length (in characters) of the requested component. If $FILESCAN does not locate the component, it returns the value 0 in this field and in the **component address** field and returns the SS$_NORMAL condition value.

item code
A user-supplied, word-length symbolic code that specifies the component desired. The $FSCNDEF macro defines the item codes. Each item code is described under "$FILESCAN Item Codes."

component address
A longword in which $FILESCAN writes the starting address of the component. This address points to a location in the input string itself.

$FILESCAN Item Codes

FSCN$_FILESPEC
When you specify FSCN$_FILESPEC, $FILESCAN returns the length and starting address of the full file specification. The full file specification may contain the node, device, directory, name, type, and version.

FSCN$_NODE
When you specify FSCN$_NODE, $FILESCAN returns the length and starting address of the node name. The node name includes the double colon (::), as well as an access control string (if present).

FSCN$_DEVICE
When you specify FSCN$_DEVICE, $FILESCAN returns the length and starting address of the device name. The device name includes the single colon (:).

FSCN$_ROOT
When you specify FSCN$_ROOT, $FILESCAN returns the length and starting address of the root directory string. The root directory name string includes the opening and closing brackets ([ ]) or angle brackets (<>).

FSCN$_DIRECTORY
When you specify FSCN$_DIRECTORY, $FILESCAN returns the length and starting address of the directory name. The directory name includes the opening and closing brackets ([ ]) or angle brackets (<>).

FSCN$_NAME
When you specify FSCN$_NAME, $FILESCAN returns the length and starting address of the file name. The file name includes no syntactical elements.

In addition, when you specify FSCN$_NAME, $FILESCAN returns the length and starting address of a quoted file specification following a node name (as in the specification NODE::"FILE-SPEC"). The beginning and ending quotation marks are included.

FSCN$_TYPE

When you specify FSCN$_TYPE, $FILESCAN returns the length and starting address of the file type. The file type includes the preceding period (.).

FSCN$_VERSION

When you specify FSCN$_VERSION, $FILESCAN returns the length and starting address of the file version number. The file version number includes the preceding period (.) or semicolon (;) delimiter.

fldflags

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

Longword flag mask in which $FILESCAN sets a bit for each file specification component found in the input string. The **fldflags** argument is the address of this longword flag mask.

The $FSCNDEF macro defines a symbolic name for each significant flag bit. The following table shows the file specification component that corresponds to the symbolic name of each flag bit.

Symbolic Name	Corresponding Component
FSCN$V_NODE	Node name
FSCN$V_DEVICE	Device name
FSCN$V_ROOT	Root directory name string
FSCN$V_DIRECTORY	Directory name
FSCN$V_NAME	File name
FSCN$V_TYPE	File type
FSCN$V_VERSION	Version number

The **fldflags** argument is optional. When you want to know which components of a file specification are present in a string, but do not need to know the contents or length of these components, you should specify **fldflags** instead of **valuelst**.

DESCRIPTION

When $FILESCAN locates a partial file specification (for example, DISK:[FOO]), it returns the length and starting address of those components that were both requested in the item list and found in the string. If a component was requested in the item list but not found in the string, $FILESCAN returns a length of 0 and starting address of 0 to the **component length** and **component address** fields of the item descriptor for that component.

The information returned about all individual components, when taken together, describes the entire contiguous file specification string. For example, to extract only the file name and file type from a full file specification string, you can add the length of these two components and use the address of the first component (file name).

The $FILESCAN service does not perform comprehensive syntax checking. Specifically, it does not check that a component has a valid length.

However, $FILESCAN does make the following checks:

- The component must have required syntactical elements; for example, a directory component must be enclosed in brackets and a node name must be followed by a double colon (::).
- The component must not contain invalid characters. Invalid characters are specific to each component. For example, a comma (,) is a valid character in a directory component but not in a file type component.
- Spaces, tabs, and carriage returns are permitted within quoted strings, but are invalid anywhere else.

Invalid characters are treated as terminators. For example, if $FILESCAN encounters a space within a file name component, it assumes that the space terminates the full file specification string.

The $FILESCAN service recognizes the DEC Multinational alphabetical characters (such as à) as alphanumeric characters.

The $FILESCAN service does not (1) assume default values for unspecified file specification components, (2) perform logical name translation on components, (3) perform wildcard processing, or (4) perform directory lookups.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The service could not read the string pointed to by the **srcstr** argument or could not write to an item descriptor in the item list specified by the **valuelst** argument.
SS$_BADPARAM	The item list contains an invalid item code.

$FIND_HELD Find Identifiers Held by User

The Find Identifiers Held by User service returns the identifier(s) held by a specified holder. When called repeatedly with a context longword, it returns in succession all the identifiers held by the specified holder.

FORMAT

SYS$FIND_HELD *holder ,[id] ,[attrib] ,[contxt]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

holder

VMS usage: **rights_holder**
type: **quadword (unsigned)**
access: **read only**
mechanism: **by reference**

Holder whose identifier(s) are to be found when $FIND_HELD completes execution. The **holder** argument is the address of a quadword data structure containing the holder identifier. This quadword data structure consists of a longword containing the holder UIC, followed by a longword containing the value zero.

id

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

Identifier value found when $FIND_HELD completes execution. The **id** argument is the address of a longword containing the identifier value with which the holder is associated.

attrib

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

Attributes associated with the identifier returned in **id** when $FIND_HELD completes execution. The **attrib** argument is the address of a longword containing a bit mask specifying the attributes.

Symbol values are offsets to the bits within the longword. You can also obtain the values as masks with the appropriate bit set using the prefix KGB$M rather than KGB$V. The symbols are defined in the system macro library ($KGBDEF). The following are the symbols for each bit position:

Bit Position	Meaning When Set
KGB$V_DYNAMIC	Allows the unprivileged holder to add or remove the identifier from the process rights list
KGB$V_RESOURCE	Allows the holder to charge resources, such as disk blocks, to the identifier

contxt

VMS usage: **context**
type: **longword (unsigned)**
access: **modify**
mechanism: **by reference**

Context value used when repeatedly calling $FIND_HELD. The **contxt** argument is the address of a longword used while searching for all identifiers. The context value must be initialized to zero, and the resulting context of each call to $FIND_HELD must be presented to each subsequent call. After **contxt** is passed to SYS$FIND_HELD, you must not modify its value.

DESCRIPTION

The Find Identifier Held by User service returns the identifier(s) associated with the specified holder. To determine all the identifiers held by the specified holder, you call SYS$FIND_HELD repeatedly until it returns the status code SS$_NOSUCHID. When SS$_NOSUCHID is returned, $FIND_HELD has returned all the identifiers, cleared the context value, and deallocated the record stream.

If you complete your calls to SYS$FIND_HELD before SS$_NOSUCHID is returned, you use SYS$FINISH_RDB to clear the context value and deallocate the record stream.

Note that when you use wildcards with this service, the records are returned in the order in which they were originally written, because the first record is located on the basis of the holder ID. Thus, all the target records have the same holder ID or, in other words, they have duplicate keys, which leads to retrieval in the order in which they were written.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The **id** argument cannot be read by the caller, or the **holder**, **attrib**, or **contxt** argument cannot be written by the caller.
SS$_IVCHAN	The contents of the contxt longword are not valid.
SS$_INSFMEM	The process dynamic memory is insufficient for opening the rights database.
SS$_IVIDENT	The specified holder identifier is of invalid format.

SS$_NOIOCHAN	No more rights database context streams are available.
SS$_NOSUCHID	The specified holder identifier does not exist, or no further identifiers are held by the specified holder.
RMS$_PRV	You do not have read access to the rights database.

Because the rights database is an indexed file accessed with VMS RMS, this service may also return RMS status codes associated with operations on indexed files. For descriptions of these status codes, refer to the *VMS Record Management Services Manual*.

$FIND_HOLDER Find Holder of Identifier

The Find Holder of Identifier service returns the holder of a specified identifier. When called repeatedly with a context longword, it returns all the holders of the specified identifier in the order in which they were added.

FORMAT

SYS$FIND_HOLDER *id ,[holder] ,[attrib] ,[contxt]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

id

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

Binary identifier value whose holders are found by $FIND_HOLDER. The **id** argument is a longword containing the binary identifier value.

holder

VMS usage: **rights_holder**
type: **quadword (unsigned)**
access: **write only**
mechanism: **by reference**

Holder identifier returned when $FIND_HOLDER completes execution. The **holder** argument is the address of a quadword containing the holder identifier. The first longword contains the UIC of the holder with the high-order word containing the group number and the low-order word containing the member number. The second longword contains the value zero.

attrib

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

Mask of attributes associated with the holder record specified by **holder**. The **attrib** argument is the address of a longword containing the attribute mask.

Symbol values are offsets to the bits within the longword. You can also obtain the values as masks with the appropriate bit set using the prefix KGB$M rather than KGB$V. The symbols are defined in the system macro library ($KGBDEF). The following are the symbols for each bit position:

Bit Position	Meaning When Set
KGB$V_DYNAMIC	Allows the unprivileged holder to add or remove the identifier from the process rights list
KGB$V_RESOURCE	Allows the holder to charge resources, such as disk blocks, to the identifier

contxt

VMS usage: **context**
type: **longword (unsigned)**
access: **modify**
mechanism: **by reference**

Context value used while searching for all the holders of the specified identifier when executing $FIND_HOLDER. The **contxt** argument is the address of a longword containing the context value. When calling $FIND_HOLDER repeatedly, **contxt** must be set initially to zero and the resulting context of each call to $FIND_HOLDER must be presented to each subsequent call. After the argument is passed to SYS$FIND_HOLDER, you must not modify its value.

DESCRIPTION

The Find Holder of Identifier service returns the holder of the specified identifier. To determine all the holders of the specified identifier, you call SYS$FIND_HOLDER repeatedly until it returns the status code SS$_NOSUCHID, which indicates that $FIND_HOLDER has returned all identifiers, cleared the context longword, and deallocated the record stream. If you complete your calls to $FIND_HOLDER before SS$_NOSUCHID is returned, you use the $FINISH_RDB service to clear the context value and deallocate the record stream.

Note that when you use wildcards with this service, the records are returned in the order in which they were originally written. (This action results from the fact that the first record is located on the basis of the identifier. Thus, all the target records have the same identifier or, in other words, they have duplicate keys, which leads to retrieval in the order in which they were written.)

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The **id** argument cannot be read by the caller, or the **holder**, **attrib**, or **contxt** argument cannot be written by the caller.
SS$_IVCHAN	The contents of the contxt longword are not valid.
SS$_INSFMEM	The process dynamic memory is insufficient for opening the rights database.

SS$_IVIDENT	The specified identifier or holder identifier is of invalid format.
SS$_NOIOCHAN	No more rights database context streams are available.
SS$_NOSUCHID	The specified identifier does not exist in the rights database, or no further holders exist for the specified identifier.
RMS$_PRV	The user does not have read access to the rights database.

Because the rights database is an indexed file accessed with VMS RMS, this service may also return RMS status codes associated with operations on indexed files. For descriptions of these status codes, refer to the *VMS Record Management Services Manual.*

$FINISH_RDB Terminate Rights Database Context

The Terminate Rights Database Context service deallocates the record stream and clears the context value used with $FIND_HELD, $FIND_HOLDER, or $IDTOASC.

FORMAT

SYS$FINISH_RDB *contxt*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENT

contxt
VMS usage: **context**
type: **longword (unsigned)**
access: **modify**
mechanism: **by reference**

Context value to be cleared when $FINISH_RDB completes execution. The **contxt** argument is a longword containing the address of the context value.

DESCRIPTION

The $FINISH_RDB service clears the context longword and deallocates the record stream associated with a sequence of rights database lookups performed by the $IDTOASC, $FIND_HOLDER, and $FIND_HELD services.

If you repeatedly call $IDTOASC, $FIND_HOLDER, or $FIND_HELD until SS$_NOSUCHID is returned, you do not need to call $FINISH_RDB because the record stream has already been deallocated and the context longword has already been cleared.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The **contxt** argument cannot be written by the caller.
SS$_IVCHAN	The contents of the contxt longword are not valid.

Because the rights database is an indexed file accessed with VMS RMS, this service may also return RMS status codes associated with operations on indexed files. For descriptions of these status codes, refer to the *VMS Record Management Services Manual.*

$FORCEX Force Exit

The Force Exit system service causes an Exit ($EXIT) service call to be issued on behalf of a specified process.

FORMAT

SYS$FORCEX *[pidadr],[prcnam],[code]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

pidadr

VMS usage: **process_id**
type: **longword (unsigned)**
access: **modify**
mechanism: **by reference**

Process identification (PID) of the process to be forced to exit. The **pidadr** argument is the address of a longword containing the PID.

The **pidadr** argument is optional, but must be specified if the process that is to be forced to exit is not in the same UIC group as the calling process.

If you specify neither the **pidadr** nor **prcnam** argument, the caller is forced to exit and control is not returned.

prcnam

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

Process name of the process that is to be forced to exit. The **prcnam** argument is the address of a character string descriptor pointing to a 1- to 15-character process name string.

The **prcnam** argument can be used only on behalf of processes in the same UIC group as the calling process. To force processes in other groups to exit, you must specify the **pidadr** argument. This restriction exists because VMS interprets the UIC group number of the calling process as part of the specified process name; the names of processes are unique to UIC groups.

If you specify neither the **pidadr** nor **prcnam** argument, the caller is forced to exit and control is not returned.

code

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

Completion code value to be used as the exit parameter. The **code** argument is a longword containing this value. If you do not specify the **code** argument, a value of 0 is passed as the completion code.

DESCRIPTION

Depending on the operation, the calling process may need a certain privilege to use $FORCEX:

- You need GROUP privilege to force an exit for a process in the same group that does not have the same UIC as the calling process.
- You need WORLD privilege to force an exit for any process in the system.

The Force Exit system service requires system dynamic memory.

The image executing in the target process follows normal exit procedures. For example, if any exit handlers have been specified, they gain control before the actual exit occurs. Use the Delete Process ($DELPRC) service if you do not want a normal exit.

When a forced exit is requested for a process, a user mode AST is queued for the target process. The AST routine causes the $EXIT service call to be issued by the target process. Because the AST mechanism is used, user mode ASTs must be enabled for the target process, or no exit occurs until ASTs are reenabled. Thus, for example, a suspended process cannot be stopped by $FORCEX. The process that calls $FORCEX receives no notification that the exit is not being performed.

The $FORCEX service completes successfully if a force exit request is already in effect for the target process but the exit is not yet completed.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The process name string or string descriptor cannot be read by the caller, or the process identification cannot be written by the caller.
SS$_IVLOGNAM	The process name string has a length equal to 0 or greater than 15.
SS$_NONEXPR	The specified process does not exist, or an invalid process identification was specified.
SS$_NOPRIV	The process does not have the privilege to force an exit for the specified process.
SS$_INSFMEM	The system dynamic memory is insufficient for the operation.

$FORMAT_ACL Format Access Control List Entry

The Format Access Control List Entry service formats the specified ACL entry (ACE) into a text string.

FORMAT

SYS$FORMAT_ACL *aclent ,[acllen] ,aclstr ,[width] ,[trmdsc] ,[indent] ,[accnam] ,[nullarg] ,[nullarg]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

aclent

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

Description of the ACE formatted when $FORMAT_ACL completes execution. The **aclent** argument is the address of a descriptor pointing to a buffer containing the description of the input ACE. The first byte of the buffer contains the length of the ACE; the second byte contains a value that identifies the type of ACE, which in turn determines the ACE format.

For more information about the ACE format, see the DESCRIPTION section.

acllen

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

Length of the output string resulting when $FORMAT_ACL completes execution. The **acllen** argument is the address of a word containing the number of characters written to **aclstr**.

aclstr

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

Formatted ACE resulting when $FORMAT_ACL completes its execution. The **aclstr** argument is the address of a string descriptor pointing to a buffer containing the output string.

width

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

Maximum width of the formatted ACE resulting when $FORMAT_ACL completes its execution. The **width** argument is the address of a word containing the maximum width of the formatted ACE. If this argument is omitted or contains zero, an infinite length display line is assumed. When the width is exceeded, the character specified by **trmdsc** is inserted.

trmdsc

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

Line termination character(s) used in the formatted ACE. The **trmdsc** argument is the address of a descriptor pointing to a character string containing the termination character(s) that are inserted for each formatted ACE when the width has been exceeded.

indent

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

Number of blank characters beginning each line of the formatted ACE. The **indent** argument is the address of a word containing the number of blank characters you want inserted at the beginning of each formatted ACE.

accnam

VMS usage: **access_bit_names**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Names of the bits in the access mask when executing the $FORMAT_ACL. The **accnam** argument is the address of an array of 32 quadword descriptors that define the names of the bits in the access mask. Each element points to the name of a bit. The first element names bit 0, the second element names bit 1, and so on. If you omit **accnam**, the following names are used:

```
Bit 0    READ
Bit 1    WRITE
Bit 2    EXECUTE
Bit 3    DELETE
Bit 4    CONTROL
Bit 5    BIT_5
Bit 6    BIT_6
   .
   .
   .
Bit 31   BIT_31
```

nullarg

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

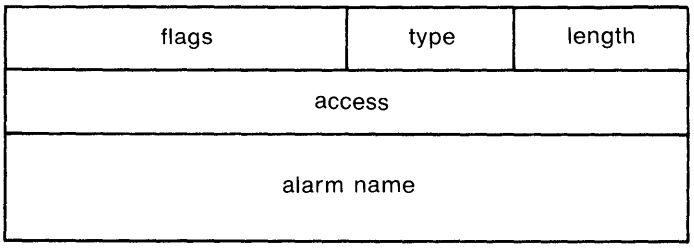
Place-holding argument reserved by DIGITAL.

DESCRIPTION

The Format Access Control List Entry service formats the specified ACL entry (ACE) into text string representation. The format for ACE type is described in the following sections. The byte offsets and type values are defined in the system macro library ($ACEDEF).

Alarm ACE

The access alarm ACE sets a security alarm. Its format is as follows:



ZK-1710-84

Field	Symbol Name	Description
length	ACE$B_SIZE	Byte containing the length in bytes of the ACE buffer
type	ACE$B_TYPE	Byte containing the type value ACE$C_ALARM
flags	ACE$W_FLAGS	Word containing alarm ACE information and ACE type-independent information
access	ACE$L_ACCESS	Longword containing a mask indicating the access modes to be watched
alarm name	ACE$T_AUDITNAME	Counted character string containing the alarm name

The flag field contains information specific to alarm ACEs and information applicable to all types of ACE. The following symbols are bit offsets to the alarm ACE information:

Bit Position	Meaning When Set
ACE$V_SUCCESS	Indicates that the alarm is raised when access is successful
ACE$V_FAILURE	Indicates that the alarm is raised when access fails

The following symbols are bit offsets to ACE information that is independent of ACE type:

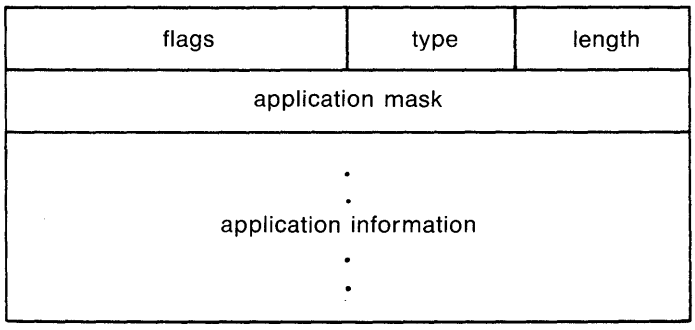
Bit Position	Meaning When Set
ACE$V_DEFAULT	This ACE is added to the ACL of any file created in the directory whose ACL contains this ACE. This option is applicable only for an ACE in a directory file's ACL.
ACE$V_HIDDEN	This ACE is application dependent. You cannot use the DCL ACL commands and the ACL editor to change the setting; the DCL command DIRECTORY/ACL does not display it.
ACE$V_NOPROPAGATE	This ACE is not propagated among versions of the same file.
ACE$V_PROTECTED	This ACE is not deleted if the entire ACL is deleted; instead you must delete this ACE explicitly.

The following symbol values are offsets to bits within the access mask. You can also obtain the symbol values as masks with the appropriate bit set using the prefix ACE$M rather than ACE$V.

Bit Position	Meaning When Set
ACE$V_READ	Read access is monitored.
ACE$V_WRITE	Write access is monitored.
ACE$V_EXECUTE	Execute access is monitored.
ACE$V_DELETE	Delete access is monitored.
ACE$V_CONTROL	Modification of the access field is monitored.

Application ACE

The application ACE contains application-dependent information. Its format is as follows:



ZK-1711-84

Field	Symbol Name	Description
length	ACE$B_SIZE	Byte containing the length in bytes of the ACE buffer.
type	ACE$B_TYPE	Byte containing the type value ACE$C_INFO.
flags	ACE$W_FLAGS	Word containing application ACE information and ACE type-independent information.
application mask	ACE$L_INFO_FLAGS	Longword containing a mask defined and used by the application.
application information	ACE$T_INFO_START	Variable length data structure defined and used by the application. The length of this data is implied by length field.

The flag field contains information specific to application ACEs and information applicable to all types of ACE. The following symbol is a bit offset to the application ACE information:

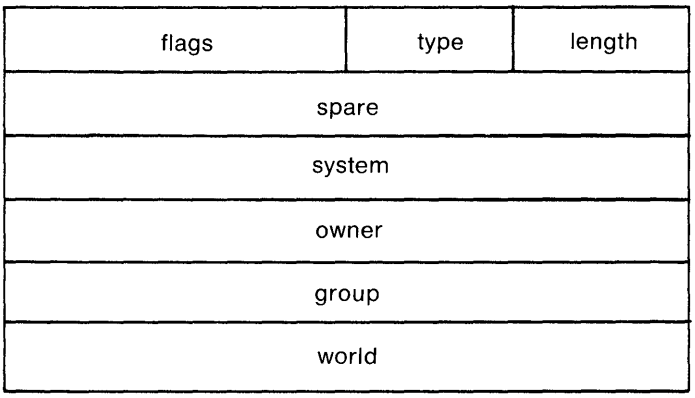
Bit Position	Meaning When Set
ACE$V_INFO_TYPE	Four-bit field containing a value indicating whether the application is a CSS application (ACE$C_CSS) or a customer application (ACE$C_CUST)

The following symbols are bit offsets to ACE information that is independent of ACE type:

Bit Position	Meaning When Set
ACE$V_DEFAULT	This ACE is added to the ACL of any file created in the directory whose ACL contains this ACE. This bit is applicable only for an ACE in a directory file's ACL.
ACE$V_HIDDEN	This bit is application dependent. You cannot use the DCL ACL commands and the ACL editor to change the setting; the DCL command DIRECTORY/ACL does not display it.
ACE$V_NOPROPAGATE	This ACE is not propagated between versions of the same file.
ACE$V_PROTECTED	This ACE is not deleted if the entire ACL is deleted; instead you must delete this ACE explicitly.

Directory Default ACE

The directory default ACE specifies the UIC-based protection for all files created in the directory. You can use this type of ACE only in the ACL of a directory file. Its format is as follows.



ZK-1712-84

Field	Symbol Name	Description
length	ACE$B_SIZE	Byte containing the length in bytes of the ACE buffer.
type	ACE$B_TYPE	Byte containing the type value ACE$C_DIRDEF.
flags	ACE$W_FLAGS	Word containing ACE type-independent information.
spare	ACE$L_SPARE1	Longword that is reserved for future use and must be zero.
system	ACE$L_SYS_PROT	Longword containing a mask indicating the access mode granted to system users. Each bit represents one type of access.
owner	ACE$L_OWN_PROT	Longword containing a mask indicating the access mode granted to the owner. Each bit represents one type of access.
group	ACE$L_GRP_PROT	Longword containing a mask indicating the access mode granted to group users. Each bit represents one type of access.
world	ACE$L_WOR_PROT	Longword containing a mask indicating the access mode granted to the world. Each bit represents one type of access.

The flag field contains information applicable to all types of ACE. The following symbols are bit offsets to ACE information that is independent of ACE type:

Bit Position	Meaning When Set
ACE$V_DEFAULT	This ACE is added to the ACL of any file created in the directory whose ACL contains this ACE. This option is applicable only for an ACE in a directory file's ACL.

Bit Position	Meaning When Set
ACE$V_HIDDEN	This ACE is application dependent. You cannot use the DCL ACL commands and the ACL editor to change the setting; the DCL command DIRECTORY/ACL does not display it.
ACE$V_NOPROPAGATE	This ACE is not propagated among versions of the same file.
ACE$V_PROTECTED	This ACE is not deleted if the entire ACL is deleted; instead you must delete this ACE explicitly.

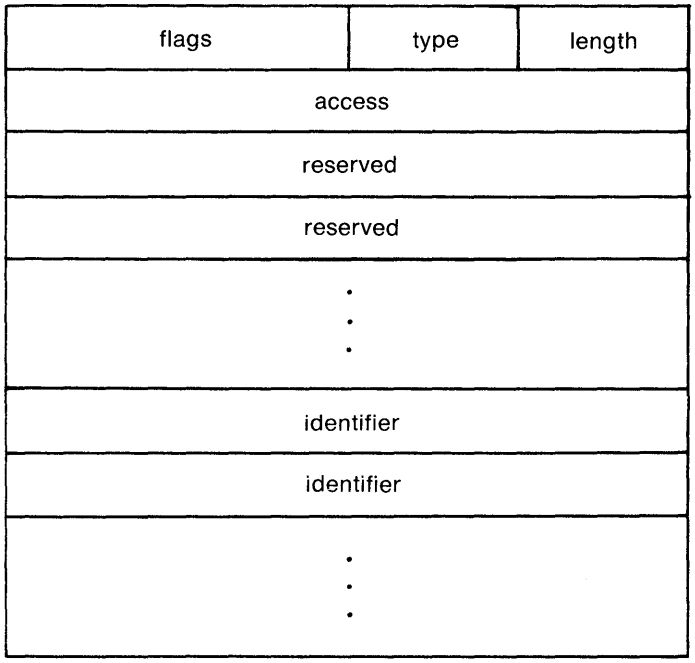
The system interprets the bits within the access mask as shown in the following table. The following symbol values are offsets to bits within the mask indicating the access mode granted in the system, owner, group, and world fields:

Bit Position	Meaning When Set
ACE$V_READ	Read access is granted.
ACE$V_WRITE	Write access is granted.
ACE$V_EXECUTE	Execute access is granted.
ACE$V_DELETE	Delete access is granted.

You can also obtain the symbol values as masks with the appropriate bit set by using the prefix ACE$M rather than ACE$V.

Identifier ACE

The identifier ACE controls access to an object based on identifiers. Its format is as follows.



ZK-1713-84

Field	Symbol Name	Description
length	ACE$B_SIZE	Byte containing the length in bytes of the ACE buffer.
type	ACE$B_TYPE	Byte containing the type value ACE$C_KEYID.
flags	ACE$W_FLAGS	Word containing identifier ACE information and ACE type-independent information.
access	ACE$L_ACCESS	Longword containing a mask indicating the access mode granted to the specified identifiers.
reserved	ACE$V_RESERVED	Longwords containing application-specific information. The number of reserved longwords is specified in the flags field.
identifier	ACE$L_KEY	Longwords containing identifiers. The number of longwords is implied by ACE$B_LENGTH. If an accessor holds all of the listed identifiers, the ACE is said to match the accessor, and the access specified in ACE$L_ACCESS is granted.

The flag field contains information specific to identifier ACEs and information applicable to all types of ACE. The following symbol is a bit offset to identifier ACE information:

Bit Position	Meaning When Set
ACE$V_RESERVED	Four-bit field containing the number of longwords to reserve for application-dependent data. The number must be between 0 and 15. The reserved longwords, if any, immediately precede the identifiers.

The following symbols are bit offsets to ACE information that is independent of ACE type:

Bit Position	Meaning When Set
ACE$V_DEFAULT	This ACE is added to the ACL of any file created in the directory whose ACL contains this ACE. This bit is applicable only for an ACE in a directory file's ACL.
ACE$V_HIDDEN	This bit is application dependent. You cannot use the DCL ACL commands and the ACL editor to change the setting; the DCL command DIRECTORY/ACL does not display it.
ACE$V_NOPROPAGATE	This ACE is not propagated between versions of the same file.
ACE$V_PROTECTED	This ACE is not deleted if the entire ACL is deleted; instead you must delete this ACE explicitly.

The following symbol values are offsets to bits within the mask indicating the access mode granted in the system, owner, group, and world fields:

Bit Position	Meaning When Set
ACE$V_READ	Read access is granted.
ACE$V_WRITE	Write access is granted.
ACE$V_EXECUTE	Execute access is granted.
ACE$V_DELETE	Delete access is granted.
ACE$V_CONTROL	Modification of the access field is granted.

You can also obtain the symbol values as masks with the appropriate bit set by using the prefix ACE$M rather than ACE$V.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The ACL entry or its descriptor cannot be read by the caller, or the string descriptor cannot be read by the caller, or the length word or the string buffer cannot be written by the caller.
SS$_BUFFEROVF	The service completed successfully. The output string has overflowed the buffer and has been truncated.

$GETDVI Get Device/Volume Information

The Get Device/Volume Information service returns information about an I/O device; this information consists of primary and secondary device characteristics.

The $GETDVI service completes asynchronously; that is, it returns to the caller after queuing the information request, without waiting for the requested information to be returned.

For synchronous completion, use the Get Device/Volume Information and Wait ($GETDVIW) service. The $GETDVIW service is identical to the $GETDVI service in every way except that $GETDVIW returns to the caller with the requested information.

For additional information about system service completion, refer to the Synchronize ($SYNCH) service and to the *Introduction to VMS System Services*.

FORMAT

SYS$GETDVI *[efn] ,[chan] ,[devnam] ,itmlst [,iosb] [,astadr] [,astprm] [,nullarg]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

efn

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

Number of the event flag to be set when $GETDVI returns the requested information. The **efn** argument is a longword containing this number; however, $GETDVI uses only the low-order byte.

Upon request initiation, $GETDVI clears the specified event flag (or event flag 0 if **efn** was not specified). Then, when $GETDVI returns the requested information, it sets the specified event flag (or event flag 0).

chan

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

Number of the I/O channel assigned to the device about which information is desired. The **chan** argument is a word containing this number.

To identify a device to $GETDVI, you can specify either the **chan** or **devnam** argument, but you should not specify both. If you specify both arguments, the **chan** argument is used.

If you specify neither **chan** nor **devnam**, $GETDVI uses a default value of 0 for **chan**.

devnam

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

The name of the device about which $GETDVI is to return information. The **devnam** argument is the address of a character string descriptor pointing to this name string.

The device name string may be either a physical device name or a logical name. If the first character in the string is an underscore (_), the string is considered a physical device name; otherwise, the string is considered a logical name and logical name translation is performed until either a physical device name is found or the system default number of translations has been performed.

If the device name string contains a colon, the colon and the characters that follow it are ignored.

To identify a device to $GETDVI, you can specify either the **chan** or **devnam** argument, but you should not specify both. If both arguments are specified, the **chan** argument is used.

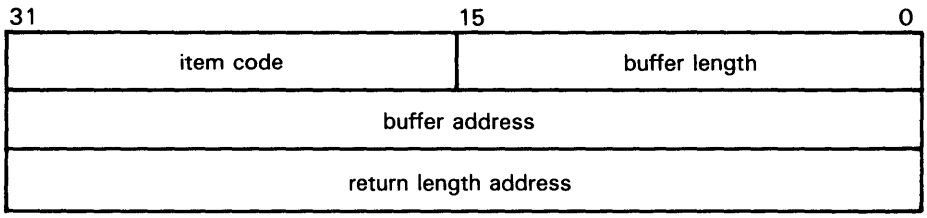
If you specify neither **chan** nor **devnam**, $GETDVI uses a default value of 0 for **chan**.

itmlst

VMS usage: **item_list_3**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Item list specifying which information about the device is to be returned. The **itmlst** argument is the address of a list of item descriptors, each of which describes an item of information. The list of item descriptors is terminated by a longword of 0.

The following diagram depicts the format of a single item descriptor.



ZK-1705-84

$GETDVI Item Descriptor Fields

buffer length

A word containing a user-supplied integer specifying the length (in bytes) of the buffer in which $GETDVI is to write the information. The length of the buffer needed depends upon the item code specified in the **item code** field of the item descriptor. If the value of **buffer length** is too small, $GETDVI truncates the data.

item code

A word containing a user-supplied symbolic code specifying the item of information that $GETDVI is to return. The $DVIDEF macro defines these codes. Each item code is described under "$GETDVI Item Codes."

buffer address

A longword containing the user-supplied address of the buffer in which $GETDVI is to write the information.

return length address

A longword containing the user-supplied address of a word in which $GETDVI writes the length in bytes of the information it returned.

$GETDVI Item Codes

DVI$_ACPPID

When you specify DVI$_ACPPID, $GETDVI returns the ACP process ID as a 4-byte hexadecimal number.

DVI$_ACPTYPE

When you specify DVI$_ACPTYPE, $GETDVI returns the ACP type code as a 4-byte hexadecimal number. The following symbols define each of the ACP type codes that $GETDVI can return.

Symbol	Description
DVI$C_ACP_F11V1	Files–11 Level 1
DVI$C_ACP_F11V2	Files–11 Level 2
DVI$C_ACP_MTA	Magnetic tape
DVI$C_ACP_NET	Networks
DVI$C_ACP_REM	Remote I/O

DVI$_ALLDEVNAM

When you specify DVI$_ALLDEVNAM, $GETDVI returns the allocation-class-device-name, which is a 64-byte hexadecimal string. The allocation-class-device-name uniquely identifies each device that is currently connected to any VAX node in a VAXcluster or to a single-node VAX. This item code generates a single unique name for a device even if the device is dual ported.

One use for the allocation-class-device-name might be in an application wherein processes need to coordinate their access to devices (not volumes) using the VMS lock manager. In this case, the program would make the device a resource to be locked by the VMS lock manager, specifying as the resource name the following concatenated components: (1) a user facility prefix followed by an underscore character and (2) the allocation-class-device-name of the device.

Note that the name returned by the DVI$_DEVLOCKNAM item code should be used to coordinate access to volumes.

DVI$_ALLOCLASS

When you specify DVI$_ALLOCLASS, $GETDVI returns the allocation class of the host as a longword integer between 0 and 255. An allocation class is a unique number between 0 and 255 that the system manager assigns to a pair of hosts and the dual-pathed devices that the hosts make available to other nodes in the VAXcluster.

The allocation class provides a way for you to access dual-pathed devices through either of the hosts that serve you to the VAXcluster. In this way, if one host of an allocation class set is not available, you can gain access to a device specified by that allocation class through the other host of the allocation class. You do not have to be concerned with which host of the allocation class provides access to the device. Specifically, the device name string is constructed of the following format:

$allocation_class$device_name

For a detailed discussion of allocation classes, refer to the *VMS VAXcluster Manual*.

DVI$_ALT_HOST_AVAIL

When you specify DVI$_ALT_HOST_AVAIL, $GETDVI returns a longword that is interpreted as Boolean. A value of 1 indicates that the host serving the alternate path is available; a value of 0 indicates that it is not.

The host is the node that makes the device available to other nodes in the VAXcluster. A host node can be either a VAX with an MSCP server or an HSC-50.

A dual-pathed device is one that is made available to the VAXcluster by two hosts. Each of the hosts provides access (serves a path) to the device for users. One host serves the primary path; the other host serves the alternate path. The primary path is the path that the system creates through the first available host.

You should not be concerned with which host provides access to the device. When accessing a device, you specify the allocation class of the desired device, not the name of the host that serves it.

If the host serving the primary path fails, the system automatically creates a path to the device through the alternate host.

DVI$_ALT_HOST_NAME

When you specify DVI$_ALT_HOST_NAME, $GETDVI returns the name of the host serving the alternate path as a 64-byte zero-filled string.

For more information about hosts, dual-pathed devices and primary and alternate paths, refer to the description of the DVI$_ALT_HOST_AVAIL item code.

DVI$_ALT_HOST_TYPE

When you specify DVI$_ALT_HOST_TYPE, $GETDVI returns, as a 4-byte string, the hardware type of the host serving the alternate path. Each hardware type has a symbolic name. The following table shows each symbolic name and the host it denotes:

Name	Host
VAX	Any VAX family processor
HS50	HSC-50
HS70	HSC-70

For more information about hosts, dual-pathed devices, and primary and alternate paths, refer to the description of the DVI$_ALT_HOST_AVAIL item code.

DVI$_CLUSTER

When you specify DVI$_CLUSTER, $GETDVI returns the volume cluster size as a 4-byte decimal number. This item code is applicable only to disks.

DVI$_CYLINDERS

When you specify DVI$_CYLINDERS, $GETDVI returns the number of cylinders on the volume as a 4-byte decimal number. This item code is applicable only to disks.

DVI$_DEVBUFSIZ

When you specify DVI$_DEVBUFSIZ, $GETDVI returns the device buffer size (for example, the width of a terminal or the block size of a tape) as a 4-byte decimal number.

DVI$_DEVCHAR

When you specify DVI$_DEVCHAR, $GETDVI returns device-independent characteristics as a 4-byte bit vector. Each characteristic is represented by a bit. When $GETDVI sets a bit, the device has the corresponding

characteristic. Each bit in the vector has a symbolic name. The $DEVDEF macro defines the following symbolic names:

Symbol	Description
DEV$V_REC	Device is record oriented.
DEV$V_CCL	Device is a carriage control device.
DEV$V_TRM	Device is a terminal.
DEV$V_DIR	Device is directory structured.
DEV$V_SDI	Device is single-directory structured.
DEV$V_SQD	Device is sequential and block oriented.
DEV$V_SPL	Device is being spooled.
DEV$V_OPR	Device is an operator.
DEV$V_RCT	Disk contains RCT; DEC standard 166 disk.
DEV$V_NET	Device is a network device.
DEV$V_FOD	Device is files oriented.
DEV$V_DUA	Device is dual ported.
DEV$V_SHR	Device is shareable.
DEV$V_GEN	Device is a generic device.
DEV$V_AVL	Device is available for use.
DEV$V_MNT	Device is mounted.
DEV$V_MBX	Device is a mailbox.
DEV$V_DMT	Device is marked for dismount.
DEV$V_ELG	Device has error logging enabled.
DEV$V_ALL	Device is allocated.
DEV$V_FOR	Device is mounted foreign.
DEV$V_SWL	Device is software write locked.
DEV$V_IDV	Device can provide input.
DEV$V_ODV	Device can provide output.
DEV$V_RND	Device allows random access.
DEV$V_RTM	Device is a real-time device.
DEV$V_RCK	Device has read-checking enabled.
DEV$V_WCK	Device has write-checking enabled.

Note that each device characteristic has its own individual $GETDVI item code with the format DVI$_xxxx, where *xxxx* are the characters following the underscore character in the symbolic name for that device characteristic.

For example, when you specify the item code DVI$_REC, $GETDVI returns a longword value that is interpreted as Boolean. If the value is 0, the device is not record oriented; if the value is 1, it is record oriented. This information is identical to that returned in the DEV$V_REC bit of the longword vector specified by the DVI$_DEVCHAR item code.

The buffer must specify a longword for all of these device-characteristic item codes.

DVI$_DEVCHAR2

When you specify DVI$_DEVCHAR2, $GETDVI returns additional device-independent characteristics as a 4-byte bit vector. Each bit in the vector, when set, corresponds to a symbolic name. The $DEVDEF macro defines these symbolic names.

DVI$_DEVCLASS

When you specify DVI$_DEVCLASS, $GETDVI returns the device class as a 4-byte decimal number. Each class has a corresponding symbol. The $DCDEF macro defines these symbols. The following table describes each device class symbol.

Symbol	Description
DC$_DISK	Disk device
DC$_TAPE	Tape device
DC$_SCOM	Synchronous communications device
DC$_CARD	Card reader
DC$_TERM	Terminal
DC$_LP	Line printer
DC$_REALTIME	Real-time
DC$_MAILBOX	Mailbox
DC$_MISC	Miscellaneous device

DVI$_DEVDEPEND

When you specify DVI$_DEVDEPEND, $GETDVI returns device-dependent characteristics as a 4-byte bit vector. To determine what information is returned for a particular device, refer to the *VMS I/O User's Reference Volume.*

Note that, for terminals only, individual $GETDVI item codes are provided for most of the informational items returned in the DVI$_DEVDEPEND longword bit vector. The names of these item codes have the format DVI$_TT_xxxx, where *xxxx* is the characteristic name. The same characteristic name follows the underscore character in the symbolic name for each bit (defined by the $TTDEF macro) in the DVI$_DEVDEPEND longword. For example, the DVI$_TT_NOECHO item code returns the same information as that returned in the DVI$_DEVDEPEND bit whose symbolic name is TT$V_NOECHO.

Each such item code requires that the buffer specify a longword value, which is interpreted as Boolean. A value of 0 indicates that the terminal does not have that characteristic; a value of 1 indicates that it does.

The list of these terminal-specific item codes follows this list of item codes.

DVI$_DEVDEPEND2

When you specify DVI$_DEVDEPEND2, $GETDVI returns additional device-dependent characteristics as a 4-byte bit vector. Refer to the *VMS I/O User's Reference Volume* to determine what information is returned for a particular device.

Note that, for terminals only, individual $GETDVI item codes are provided for most of the informational items returned in the DVI$_DEVDEPEND2 longword bit vector. As with DVI$_DEVDEPEND, the same characteristic name appears in the item code as appears in the symbolic name defined for each bit in the DVI$_DEVDEPEND2 longword, except that in the case of DVI$_DEVDEPEND2, the symbolic names for bits are defined by the $TT2DEF macro.

The list of these terminal-specific item codes follows this list of item codes.

DVI$_DEVLOCKNAM

When you specify DVI$_DEVLOCKNAM, $GETDVI returns the device lock name, which is a 64-byte hexadecimal string. The device lock name uniquely identifies each volume or volume set in a VAXcluster or in a single node VAX. This item code is applicable only to disks.

The item code is applicable to all disk volumes and volume sets: mounted, not mounted, mounted shared, mounted private, or mounted foreign.

The device lock name is assigned to a volume when it is first mounted, and you cannot change this name, even if the volume name itself is changed. This allows any process on any VAX node in a VAXcluster to access a uniquely identified volume.

One use for the device lock name might be in an application wherein processes need to coordinate their access to files using the VMS lock manager. In this case, the program would make the file a resource to be locked by the VMS lock manager, specifying as the resource name the following concatenated components: (1) a user facility prefix followed by an underscore character, (2) the device lock name of the volume on which the file resides, and (3) the file ID of the file.

DVI$_DEVNAM

When you specify DVI$_DEVNAM, $GETDVI returns the device name as a 64-byte, zero-filled string. The node name is not returned.

DVI$_DEVSTS

When you specify DVI$_DEVSTS, $GETDVI returns device-dependent status information as a 4-byte bit vector. The $UCBDEF macro defines symbols for the status bits. For this device-dependent information, refer to the *VMS I/O User's Reference Volume.*

DVI$_DEVTYPE

When you specify DVI$_DEVTYPE, $GETDVI returns the device type as a 4-byte decimal number. The $DCDEF macro defines symbols for the device types.

DVI$_DISPLAY_DEVNAM

When you specify DVI$_DISPLAY_DEVNAM, $GETDVI returns the preferred device name for user displays as a 256-byte zero-filled string. The DVI$_DISPLAY_DEVNAM item code is not recommended for use with

the $ASSIGN service. Use the DVI4_ALLDEVNAM item code to return an allocation class device name that is usable as input to a program.

DVI$_ERRCNT

When you specify DVI$_ERRCNT, $GETDVI returns the device's error count as a 4-byte decimal number.

DVI$_FREEBLOCKS

When you specify DVI$_FREEBLOCKS, $GETDVI returns the number of free blocks on a disk as a 4-byte decimal number. This item code is applicable only to disks.

DVI$_FULLDEVNAM

When you specify DVI$_FULLDEVNAM, $GETDVI returns the node name and device name as a 64-byte, zero-filled string.

The DVI$_FULLDEVNAM item code is useful in a VAXcluster environment because, unlike DVI$_DEVNAM, DVI$_FULLDEVNAM returns the name of the VAX node on which the device resides.

One use for the DVI$_FULLDEVNAM item code might be to retrieve the name of a device in order to have that name displayed on a terminal. However, you should not use this name as a resource name as input to the lock manager; use the name returned by the DVI$_DEVLOCKNAM item code for locking volumes and the name returned by DVI$_ALLDEVNAM for locking devices.

DVI$_HOST_AVAIL

When you specify DVI$_HOST_AVAIL, $GETDVI returns a longword, which is interpreted as Boolean. A value of 1 indicates that the host serving the primary path is available; a value of 0 indicates that it is not.

For more information about hosts, dual-pathed devices, and primary and alternate paths, refer to the description of the DVI$_ALT_HOST_AVAIL item code.

DVI$_HOST_COUNT

When you specify DVI$_HOST_COUNT, $GETDVI returns, as a longword integer, the number of hosts that make the device available to other nodes in the VAXcluster. One or two hosts, but no more, can make a device available to other nodes in the VAXcluster.

For more information about hosts, dual-pathed devices, and primary and alternate paths, refer to the description of the DVI$_ALT_HOST_AVAIL item code.

DVI$_HOST_NAME

When you specify DVI$_HOST_NAME, $GETDVI returns (as a 64-byte, zero-filled string) the name of the host serving the primary path.

For more information about hosts, dual-pathed devices, and primary and alternate paths, refer to the description of the DVI$_ALT_HOST_AVAIL item code.

DVI$_HOST_TYPE

When you specify DVI$_HOST_TYPE, $GETDVI returns, as a 4-byte string, the type of host serving the primary path. Each hardware type has a

symbolic name. The following table shows each symbolic name and the host it denotes.

Name	Host
VAX	Any VAX family processor
HS50	HSC-50
HS70	HSC-70

For more information about hosts, dual-pathed devices, and primary and alternate paths, refer to the description of the DVI$_ALT_HOST_AVAIL item code.

DVI$_LOCKID

When you specify DVI$_LOCKID, $GETDVI returns the lock ID of the lock on a disk. The VMS lock manager locks a disk if it is available to all VAX nodes in a VAXcluster and it is either allocated or mounted. A disk is available to all VAX nodes in a VAXcluster if, for example, it is served by an HSC controller or MSCP server, or if it is a dual-ported MASSBUS disk.

The buffer must specify a longword into which $GETDVI is to return the 4-byte hexadecimal lock ID.

DVI$_LOGVOLNAM

When you specify DVI$_LOGVOLNAM, $GETDVI returns the logical name of the volume or volume set as a 64-byte string.

DVI$_MAXBLOCK

When you specify DVI$_MAXBLOCK, $GETDVI returns the maximum number of blocks on the volume as a 4-byte decimal number. This item code is applicable only to disks.

DVI$_MAXFILES

When you specify DVI$_MAXFILES, $GETDVI returns the maximum number of files on the volume as a 4-byte decimal number. This item code is applicable only to disks.

DVI$_MEDIA_ID

When you specify DVI$_MEDIA_ID, $GETDVI returns the nondecoded media ID as a longword. This item code is applicable only to disks and tapes.

DVI$_MEDIA_NAME

When you specify DVI$_MEDIA_NAME, $GETDVI returns the name of the volume type (for example, RK07 or TA78) as a 64-byte, zero-filled string. This item code is applicable only to disks and tapes.

DVI$_MEDIA_TYPE

When you specify DVI$_MEDIA_TYPE, $GETDVI returns the device name prefix of the volume (for example, DM for an RK07 device or MU for a TA78 device) as a 64-byte, zero-filled string. This item code is applicable only to disks and tapes.

DVI$_MOUNTCNT

When you specify DVI$_MOUNTCNT, $GETDVI returns the mount count for the volume as a 4-byte decimal number.

DVI$_MSCP_UNIT_NUMBER
When you specify DVI$_MSCP_UNIT_NUMBER, $GETDVI returns the internal coded value for MSCP unit numbers as a longword integer. This item code is reserved by DIGITAL.

DVI$_NEXTDEVNAM
When you specify DVI$_NEXTDEVNAM, $GETDVI returns the device name of the next volume in the volume set as a 64-byte, zero-filled string. This item code is applicable only to disks.

DVI$_OPCNT
When you specify DVI$_OPCNT, $GETDVI returns the operation count for the volume as a 4-byte decimal number.

DVI$_OWNUIC
When you specify DVI$_OWNUIC, $GETDVI returns the user identification code (UIC) of the owner of the device as a standard 4-byte VMS UIC.

DVI$_PID
When you specify DVI$_PID, $GETDVI returns the process identification (PID) of the owner of the device as a 4-byte hexadecimal number.

DVI$_RECSIZ
When you specify DVI$_RECSIZ, $GETDVI returns the blocked record size as a 4-byte decimal number.

DVI$_REFCNT
When you specify DVI$_REFCNT, $GETDVI returns the number of channels assigned to the device as a 4-byte decimal number.

DVI$_REMOTE_DEVICE
When you specify DVI$_REMOTE_DEVICE, $GETDVI returns a longword, which is interpreted as Boolean. A value of 1 indicates that the device is a remote device; a value of 0 indicates that it is not a remote device. A remote device is a device which is not directly connected to the local node, but instead is visible through the VAXcluster.

DVI$_ROOTDEVNAM
When you specify DVI$_ROOTDEVNAM, $GETDVI returns the device name of the root volume in the volume set as a 64-byte, zero-filled string. This item code is applicable only to disks.

DVI$_SECTORS
When you specify DVI$_SECTORS, $GETDVI returns the number of sectors per track as a 4-byte decimal number. This item code is applicable only to disks.

DVI$_SERIALNUM
When you specify DVI$_SERIALNUM, $GETDVI returns the serial number of the volume as a 4-byte decimal number. This item code is applicable only to disks.

DVI$_SERVED_DEVICE

When you specify DVI$_SERVED_DEVICE, $GETDVI returns a longword, which is interpreted as Boolean. A value of 1 indicates that the device is a served device; a value of 0 indicates that it is not a served device. A served device is one whose local node makes it available to other nodes in the VAXcluster.

DVI$_SHDW_CATCHUP_COPYING

This item code applies only to the volume shadowing option. See the *VAX Volume Shadowing Manual.*

DVI$_SHDW_FAILED_MEMBER

This item code applies only to the volume shadowing option. See the *VAX Volume Shadowing Manual.*

DVI$_SHDW_MASTER

This item code applies only to the volume shadowing option. See the *VAX Volume Shadowing Manual.*

DVI$_SHDW_MASTER_NAME

This item code applies only to the volume shadowing option. See the *VAX Volume Shadowing Manual.*

DVI$_SHDW_MEMBER

This item code applies only to the volume shadowing option. See the *VAX Volume Shadowing Manual.*

DVI$_SHDW_MERGE_COPYING

This item code applies only to the volume shadowing option. See the *VAX Volume Shadowing Manual.*

DVI$_SHDW_NEXT_MBR_NAME

This item code applies only to the volume shadowing option. See the *VAX Volume Shadowing Manual.*

DVI$_STS

When you specify DVI$_STS, $GETDVI returns the device unit status as a 4-byte bit vector. Each bit in the vector, when set, corresponds to a symbolic name that is defined by the $UCBDEF macro. The following table describes each name.

Symbol	Description
UCB$V_TIM	Time out is enabled.
UCB$V_INT	Interrupt is expected.
UCB$V_ERLOGIP	Error log is in progress on unit.
UCB$V_CANCEL	I/O on unit is cancelled.
UCB$V_ONLINE	Unit is on line.
UCB$V_POWER	Power failed while unit busy.
UCB$V_TIMOUT	Unit timed out.
UCB$V_INTTYPE	Receiver interrupt.

Symbol	Description
UCB$V_BSY	Unit is busy.
UCB$V_MOUNTING	Device is being mounted.
UCB$V_DEADMO	Deallocate at dismount.
UCB$V_VALID	Volume is software valid.
UCB$V_UNLOAD	Unload volume at dismount.
UCB$V_TEMPLATE	Template UCB from which other UCBs for this device type are made.
UCB$V_MNTVERIP	Mount verification is in progress.
UCB$V_WRONGVOL	Wrong volume detected during mount verification.
UCB$V_DELETEUCB	Delete this UCB when reference count equals 0.

DVI$_TRACKS
When you specify DVI$_TRACKS, $GETDVI returns the number of tracks per cylinder as a 4-byte decimal number. This item code is applicable only to disks.

DVI$_TRANSCNT
When you specify DVI$_TRANSCNT, $GETDVI returns the transaction count for the volume as a 4-byte decimal number.

DVI$_TT_ACCPORNAM
When you specify DVI$_TT_ACCPORNAM, $GETDVI returns the name of the remote access port associated with a channel number, or a physical or virtual terminal device number. If you specify a device which is not a remote terminal, or a remote type that does not support this feature, $GETDVI returns a null string. The $GETDVI service returns the access port name as a 64-byte zero-fill string.

The $GETDVI service returns the name in the format of the remote system. If the remote system is a LAT terminal server, $GETDVI returns the name as *server_name/port_name*. The names are separated by the slash (/) character. If the remote system is an X.29 (VAX PSI) terminal, the name is returned as *network.remote_DTE*.

When writing applications, you should use the string returned by DVI$_ACCPORNAM, instead of the physical device name, to identify remote terminals.

DVI$_TT_PHYDEVNAM
When you specify DVI$_TT_PHYDEVNAM, $GETDVI returns a string containing the physical device name of a terminal. If the caller specifies a disconnected virtual terminal, or a device that is not a terminal, $GETDVI returns a null string. $GETDVI returns the physical device name as a 64-byte zero-filled string.

DVI$_UNIT
When you specify DVI$_UNIT, $GETDVI returns the unit number as a 4-byte decimal number.

DVI$_VOLCOUNT
When you specify DVI$_VOLCOUNT, $GETDVI returns the number of volumes in the volume set as a 4-byte decimal number. This item code is applicable only to disks.

DVI$_VOLNAM
When you specify DVI$_VOLNAM, $GETDVI returns the volume name as a 12-byte zero-filled string.

DVI$_VOLNUMBER
When you specify DVI$_VOLNUMBER, $GETDVI returns the volume number of this volume in the volume set as a 4-byte decimal number. This item code is applicable only to disks.

DVI$_VOLSETMEM
When you specify DVI$_VOLSETMEM, $GETDVI returns a longword value, which is interpreted as Boolean. A value of 1 indicates that the device is part of a volume set; a value of 0 indicates that it is not. This item code is applicable only to disks.

DVI$_VPROT
When you specify DVI$_VPROT, $GETDVI returns the volume protection mask as a standard 4-byte protection mask.

DVI$_TT_xxxx
DVI$_TT_xxxx is the format for a series of item codes that return information about terminals. This information consists of terminal characteristics. The *xxxx* portion of the item code name specifies a single terminal characteristic.

Each of these item codes requires that the buffer specify a longword into which $GETDVI will write a 0 or 1: 0 if the terminal does not have the specified characteristic, and 1 if the terminal does have it. The one exception is the DVI$_TT_PAGE item code, which when specified causes $GETDVI to return a decimal longword value that is the page size of the terminal.

You can also obtain this terminal-specific information by using the DVI$_DEVDEPEND and DVI$_DEVDEPEND2 item codes. Each of these two item codes specifies a longword bit vector wherein each bit corresponds to a terminal characteristic; $GETDVI sets the corresponding bit for each characteristic possessed by the terminal.

Following is a list of the item codes that return information about terminal characteristics. For information about these characteristics, refer to the description of the F$GETDVI lexical function in the *VMS DCL Dictionary*.

Terminal-Specific $GETDVI Item Codes

DVI$_TT_NOECHO	DVI$_TT_NOTYPEAHD
DVI$_TT_HOSTSYNC	DVI$_TT_TTSYNC
DVI$_TT_ESCAPE	DVI$_TT_LOWER
DVI$_TT_MECHTAB	DVI$_TT_WRAP
DVI$_TT_LFFILL	DVI$_TT_SCOPE
DVI$_TT_CRFILL	DVI$_TT_SETSPEED

Terminal-Specific $GETDVI Item Codes	
DVI$_TT_EIGHTBIT	DVI$_TT_MBXDSABL
DVI$_TT_READSYNC	DVI$_TT_MECHFORM
DVI$_TT_NOBRDCST	DVI$_TT_HALFDUP
DVI$_TT_MODEM	DVI$_TT_OPER
DVI$_TT_LOCALECHO	DVI$_TT_AUTOBAUD
DVI$_TT_PAGE	DVI$_TT_HANGUP
DVI$_TT_MODHANGUP	DVI$_TT_BRDCSTMBX
DVI$_TT_DMA	DVI$_TT_ALTYPEAHD
DVI$_TT_ANSICRT	DVI$_TT_REGIS
DVI$_TT_AVO	DVI$_TT_EDIT
DVI$_TT_BLOCK	DVI$_TT_DECCRT
DVI$_TT_EDITING	DVI$_TT_INSERT
DVI$_TT_DIALUP	DVI$_TT_SECURE
DVI$_TT_FALLBACK	DVI$_TT_DISCONNECT
DVI$_TT_PASTHRU	DVI$_TT_SIXEL
DVI$_TT_PRINTER	DVI$_TT_APP_KEYPAD
DVI$_TT_DRCS	DVI$_TT_SYSPWD
DVI$_TT_DECCRT2	

DVI$_yyyy

DVI$_yyyy is the format for a series of item codes that return device-independent characteristics of a device. There is an item code for each device characteristic returned in the longword bit vector specified by the DVI$_DEVCHAR item code.

In the description of the DVI$_DEVCHAR item code is a list of symbol names, in which each symbol represents a device characteristic. To construct the $GETDVI item code for each device characteristic, substitute for *yyyy* that portion of the symbol name that follows the underscore character. For example, the DVI$_REC item code returns the same information as the DEV$V_REC bit in the DVI$_DEVCHAR longword bit vector.

The buffer for each of these item codes must specify a longword value, which is interpreted as Boolean. The $GETDVI service writes the value 1 into the longword, if the device has the specified characteristic, and the value 0 if it does not.

iosb

VMS usage: **io_status_block**
type: **quadword (unsigned)**
access: **write only**
mechanism: **by reference**

I/O status block that is to receive the final completion status. The **iosb** argument is the address of the quadword I/O status block.

When you specify the **iosb** argument, $GETDVI sets the quadword to zero upon request initiation. Upon request completion, a condition value is returned to the first longword; the second longword is reserved by DIGITAL.

Though this argument is optional, DIGITAL strongly recommends that you specify it, for the following reasons:

- If you are using an event flag to signal the completion of the service, you can test the I/O status block for a condition value to be sure that the event flag was not set by an event other than service completion.
- If you are using the $SYNCH service to synchronize completion of the service, the I/O status block is a required argument for $SYNCH.
- The condition value returned in R0 and the condition value returned in the I/O status block provide information about different aspects of the call to the $GETDVI service. The condition value returned in R0 gives you information about the success or failure of the service call itself; the condition value returned in the I/O status block gives you information about the success or failure of the service operation. Therefore, to accurately assess the success or failure of the call to $GETDVI, you must check the condition values returned in both R0 and the I/O status block.

Refer to the *Introduction to VMS System Services* for more information about system service completion.

astadr

VMS usage: **ast_procedure**
type: **procedure entry mask**
access: **call without stack unwinding**
mechanism: **by reference**

AST service routine to be executed when $GETDVI completes. The **astadr** argument is the address of the entry mask of this routine.

If you specify **astadr**, the AST routine executes at the same access mode as the caller of the $GETDVI service.

astprm

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

AST parameter to be passed to the AST service routine specified by the **astadr** argument. The **astprm** argument is the longword parameter.

nullarg

VMS usage: **null_arg**
type: **quadword (unsigned)**
access: **read only**
mechanism: **by reference**

Place-holding argument reserved by DIGITAL.

DESCRIPTION

You can use the **chan** argument only if (1) the channel has already been assigned, and (2) the caller's access mode is equal to or more privileged than the access mode from which the original channel assignment was made.

The caller of $GETDVI does not need to have a channel assigned to the device about which information is desired.

The $GETDVI service returns information about both primary device characteristics and secondary device characteristics. By default, $GETDVI returns information about the primary device characteristics only.

To obtain information about secondary device characteristics, you must OR the item code specifying the information desired with the code DVI$C_SECONDARY.

You can obtain information about primary and secondary devices in a single call to $GETDVI.

In most cases, the two sets of characteristics (primary and secondary) returned by $GETDVI 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 (such as the disk) and the secondary characteristics are those of the spooled device (such as the printer).
- If the device represents a logical link on the network, the secondary characteristics contain information about the link.

Unless otherwise stated in the description of the item code, $GETDVI returns information about the local node only.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The device name string descriptor, device name string, or **itmlst** argument cannot be read; or the buffer or return length longword cannot be written by the caller.
SS$_BADPARAM	The item list contains an invalid item code, or the **return length address** field in an item descriptor specifies less than four bytes for the return length information.
SS$_EXASTLM	The process has exceeded its AST limit quota.
SS$_IVCHAN	You specified an invalid channel number, that is, a channel number larger than the number of channels.
SS$_IVDEVNAM	The device name string contains invalid characters, or neither the **devnam** nor **chan** argument was specified.
SS$_IVLOGNAM	The device name string has a length of 0 or has more than 63 characters.
SS$_NONLOCAL	The device is on a remote system.
SS$_NOPRIV	The specified channel is not assigned or was assigned from a more privileged access mode.
SS$_NOSUCHDEV	The specified device does not exist on the host system.

CONDITION VALUES RETURNED IN THE I/O STATUS BLOCK

Same as those returned in R0.

$GETDVIW Get Device/Volume Information and Wait

The Get Device/Volume Information and Wait service returns information about an I/O device; this information consists of primary and secondary device characteristics.

The $GETDVIW service completes synchronously; that is, it returns to the caller with the requested information.

For asynchronous completion, use the Get Device/Volume Information ($GETDVI) service; $GETDVI returns to the caller after queuing the information request, without waiting for the information to be returned. In all other respects, $GETDVIW is identical to $GETDVI. For all other information about the $GETDVIW service, refer to the documentation of $GETDVI.

For additional information about system service completion, refer to the Synchronize ($SYNCH) service and to the *Introduction to VMS System Services*.

FORMAT

SYS$GETDVIW *[efn] ,[chan] ,[devnam] ,itmlst [,iosb] [,astadr] [,astprm] [,nullarg]*

$GETJPI Get Job/Process Information

The Get Job/Process Information service returns information about one or more processes on the system.

The $GETJPI service completes asynchronously; that is, it returns to the caller after queuing the information request, without waiting for the requested information to be returned.

For synchronous completion, you use the Get Job/Process Information and Wait ($GETJPIW) service. The $GETJPIW service is identical to the $GETJPI service in every way except that $GETJPIW returns to the caller with the requested information.

For additional information about system service completion, refer to the Synchronize ($SYNCH) service and to the *Introduction to VMS System Services*.

FORMAT

SYS$GETJPI *[efn] ,[pidadr] ,[prcnam] ,itmlst [,iosb] [,astadr] [,astprm]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

efn

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

Number of the event flag to be set when $GETJPI returns the requested information. The **efn** argument is a longword containing this number; however, $GETJPI uses only the low-order byte.

Upon request initiation, $GETJPI clears the specified event flag (or event flag 0 if **efn** was not specified). Then, when $GETJPI returns the requested information, it sets the specified event flag (or event flag 0).

pidadr

VMS usage: **process_id**
type: **longword (unsigned)**
access: **modify**
mechanism: **by reference**

Process identification (PID) of the process about which $GETJPI is to return information. The **pidadr** argument is the address of a longword containing the PID.

If you specify **pidadr** as –1, $GETJPI assumes a wildcard operation and returns the requested information for each process on the system that it has the privilege to access, one process per call. To perform a wildcard operation, you must call $GETJPI in a loop, testing for the condition value SS$_NOMOREPROC after each call and exiting the loop when SS$_NOMOREPROC is returned.

For more information, see the *Introduction to VMS System Services.*

prcnam

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

Name of the process about which $GETJPI is to return information. The **prcnam** argument is the address of a character string descriptor pointing to this name string.

The process name string must contain from 1 to 15 characters and must correspond exactly to the process name; no trailing blanks or abbreviations are permitted.

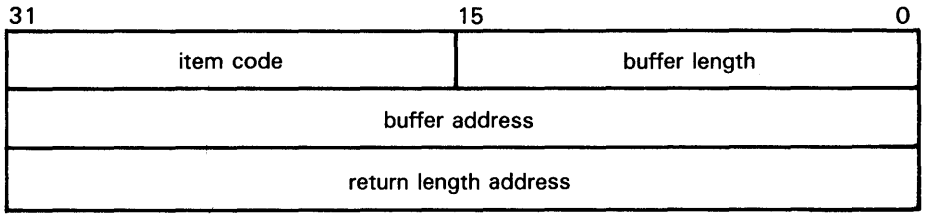
You may use the **prcnam** argument only if the process identified by **prcnam** has the same UIC group number as the calling process. If the process has a different group number, $GETJPI returns no information. To obtain information about processes in other groups, you must use the **pidadr** argument.

itmlst

VMS usage: **item_list_3**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Item list specifying which information about the process or processes is to be returned. The **itmlst** argument is the address of a list of item descriptors, each of which describes an item of information. The list of item descriptors is terminated by a longword of 0.

The following diagram depicts the format of a single item descriptor.



ZK-1705-84

$GETJPI Item Descriptor Fields

buffer length

A word containing a user-supplied integer specifying the length (in bytes) of the buffer in which $GETJPI is to write the information. The length of the buffer needed depends upon the item code specified in the **item code** field of the item descriptor. If the value of **buffer length** is too small, $GETJPI truncates the data.

item code

A word containing a user-supplied symbolic code specifying the item of information that $GETJPI is to return. The $JPIDEF macro defines these codes. Each item code is described after this list of item descriptor fields.

buffer address

A longword containing the user-supplied address of the buffer in which $GETJPI is to write the information.

return length address

A longword containing the user-supplied address of a word in which $GETJPI writes the length in bytes of the information it actually returned.

$GETJPI Item Codes

JPI$_ACCOUNT

When you specify JPI$_ACCOUNT, $GETJPI returns the account name of the process, which is an 8-byte string, filled with trailing blanks if necessary.

JPI$_APTCNT

When you specify JPI$_APTCNT, $GETJPI returns the active page table count of the process, which is a longword integer value.

JPI$_ASTACT

When you specify JPI$_ASTACT, $GETJPI returns the names of the access modes having active ASTs. This information is returned in a longword bit vector. When bit 0 is set, an active kernel-mode AST exists; bit 1, an executive-mode AST; bit 2, a supervisor-mode AST; and bit 3, a user-mode AST.

JPI$_ASTCNT

When you specify JPI$_ASTCNT, $GETJPI returns a count of the remaining AST quota, which is a longword integer value.

JPI$_ASTEN
When you specify JPI$_ASTEN, $GETJPI returns the names of the access modes having ASTs enabled. This information is returned in a longword bit vector. When bit 0 is set, kernel mode has ASTs enabled; bit 1, executive mode; bit 2, supervisor mode; and bit 3, user mode.

JPI$_ASTLM
When you specify JPI$_ASTLM, $GETJPI returns the AST limit quota of the process, which is a longword integer value.

JPI$_AUTHPRI
When you specify JPI$_AUTHPRI, $GETJPI returns the authorized base priority of the process, which is a longword integer value. The authorized base priority is the highest priority a process without ALTPRI privilege can attain by means of the $SETPRI service.

JPI$_AUTHPRIV
When you specify JPI$_AUTHPRIV, $GETJPI returns the privileges that the process is authorized to enable. These privileges are returned in a quadword privilege mask and are defined by the $PRVDEF macro.

JPI$_BIOCNT
When you specify JPI$_BIOCNT, $GETJPI returns a count of the remaining buffered I/O quota, which is a longword integer value.

JPI$_BIOLM
When you specify JPI$_BIOLM, $GETJPI returns the buffered I/O limit quota of the process, which is a longword integer value.

JPI$_BUFIO
When you specify JPI$_BUFIO, $GETJPI returns a count of the buffered I/O operations of the process, which is a longword integer value.

JPI$_BYTCNT
When you specify JPI$_BYTCNT, $GETJPI returns the remaining buffered I/O byte count quota of the process, which is a longword integer value.

JPI$_BYTLM
When you specify JPI$_BYTLM, $GETJPI returns the buffered I/O byte count limit quota of the process, which is a longword integer value.

JPI$_CHAIN
When you specify JPI$_CHAIN, $GETJPI processes another item list immediately after processing the current one. The **buffer address** field in the item descriptor specifies the address of the next item list to be processed. You must specify the JPI$_CHAIN item code last in the item list.

JPI$_CLINAME
When you specify JPI$_CLINAME, $GETJPI returns the name of the command language interpreter that the process is currently using. Because the CLI name can include up to 39 characters, the **buffer length** field in the item descriptor should specify 39 bytes.

JPI$_CPULIM
When you specify JPI$_CPULIM, $GETJPI returns the CPU time limit of the process, which is a longword integer value.

JPI$_CPUTIM
When you specify JPI$_CPUTIM, $GETJPI returns the process's accumulated CPU time in 10-millisecond ticks, which is a longword integer value.

JPI$_CREPRC_FLAGS
When you specify JPI$_CREPRC_FLAGS, $GETJPI returns the flags specified by the **stsflg** argument in the $CREPRC call that created the process. The flags are returned as a longword bit vector.

JPI$_CURPRIV
When you specify JPI$_CURPRIV, $GETJPI returns the current privileges of the process. These privileges are returned in a quadword privilege mask and are defined by the $PRVDEF macro.

JPI$_DFPFC
When you specify JPI$_DFPFC, $GETJPI returns the default page fault cluster size of the process, which is a longword integer value.

JPI$_DFWSCNT
When you specify JPI$_DFWSCNT, $GETJPI returns the default working set size of the process, which is a longword integer value.

JPI$_DIOCNT
When you specify JPI$_DIOCNT, $GETJPI returns the remaining direct I/O quota of the process, which is a longword integer value.

JPI$_DIOLM
When you specify JPI$_DIOLM, $GETJPI returns the direct I/O quota limit of the process, which is a longword integer value.

JPI$_DIRIO
When you specify JPI$_DIRIO, $GETJPI returns a count of the direct I/O operations of the process, which is a longword integer value.

JPI$_EFCS
When you specify JPI$_EFCS, $GETJPI returns the state of the process's local event flags 0 through 31 as a longword bit vector.

JPI$_EFCU
When you specify JPI$_EFCU, $GETJPI returns the state of the process's local event flags 32 through 63 as a longword bit vector.

JPI$_EFWM
When you specify JPI$_EFWM, $GETJPI returns the event flag wait mask of the process, which is a longword bit vector.

JPI$_ENQCNT
When you specify JPI$_ENQCNT, $GETJPI returns the remaining lock request quota of the process, which is a longword integer value.

JPI$_ENQLM
When you specify JPI$_ENQLM, $GETJPI returns the lock request quota of the process, which is a longword integer value.

JPI$_EXCVEC
When you specify JPI$_EXCVEC, $GETJPI returns the address of a list of exception vectors for the process. Each exception vector in the list is a longword. There are eight vectors in the list: these are, in order, a primary and a secondary vector for kernel-mode access, for executive-mode access, for supervisor-mode access, and for user-mode access.

The $GETJPI service cannot return this information for any process other than the calling process; if you specify this item code and the process is not the calling process, $GETJPI returns a zero in the buffer.

JPI$_FILCNT
When you specify JPI$_FILCNT, $GETJPI returns the remaining open file quota of the process, which is a longword integer value.

JPI$_FILLM
When you specify JPI$_FILLM, $GETJPI returns the open file limit quota of the process, which is a longword value.

JPI$_FINALEXC
When you specify JPI$_FINALEXC, $GETJPI returns the address of a list of final exception vectors for the process. Each exception vector in the list is a longword. There are four vectors in the list, one for each access mode, in the order kernel, executive, supervisor, and user.

The $GETJPI service cannot return this information for any process other than the calling process; if you specify this item code and the process is not the calling process, $GETJPI returns a zero in the buffer.

JPI$_FREP0VA
When you specify JPI$_FREP0VA, $GETJPI returns the address of the first free page at the end of the program region (P0 space) of the process.

JPI$_FREP1VA
When you specify JPI$_FREP1VA, $GETJPI returns the address of the first free page at the end of the control region (P1 space) of the process.

JPI$_FREPTECNT
When you specify JPI$_FREPTECNT, $GETJPI returns the number of pages that the process has available for virtual memory expansion. This value is a longword integer value.

JPI$_GPGCNT
When you specify JPI$_GPGCNT, $GETJPI returns the process's global page count in the working set, which is a longword integer value.

JPI$_GRP
When you specify JPI$_GRP, $GETJPI returns the group number of the process's UIC. This is a longword integer value.

JPI$_IMAGECOUNT
When you specify JPI$_IMAGECOUNT, $GETJPI returns, as a longword integer value, the number of images that have been run down for the process.

JPI$_IMAGNAME
When you specify JPI$_IMAGNAME, $GETJPI returns, as a character string, the directory specification and the image file name.

JPI$_IMAGPRIV
When you specify JPI$_IMAGPRIV, $GETJPI returns a quadword mask of the privileges with which the current image was installed. If the current image was not installed, $GETJPI returns a zero in the buffer.

JPI$_JOBPRCCNT
When you specify JPI$_JOBPRCCNT, $GETJPI returns the total number of subprocesses owned by the job, which is a longword integer value.

JPI$_JOBTYPE
When you specify JPI$_JOBTYPE, $GETJPI returns the execution mode of the process at the root of the job tree, which is a longword integer value. The symbolic name and value for each execution mode are listed in the following table. The $JPIDEF macro defines the symbolic names.

Mode Name	Value
JPI$K_DETACHED	0
JPI$K_NETWORK	1
JPI$K_BATCH	2
JPI$K_LOCAL	3
JPI$K_DIALUP	4
JPI$K_REMOTE	5

JPI$_LOGINTIM
When you specify JPI$_LOGINTIM, $GETJPI returns the time at which the process was created, which is a standard 64-bit absolute time.

JPI$_MASTER_PID
When you specify JPI$_MASTER_PID, $GETJPI returns the process identification (PID) of the master process in the job. The PID is a longword hexadecimal value.

JPI$_MAXDETACH
When you specify JPI$_MAXDETACH, $GETJPI returns the maximum number of detached processes allowed for the user who owns the process specified in the call to $GETJPI. This limit is set in the UAF record of the user. The number is returned as a word decimal value. A value of 0 means that there is no limit on the number of detached processes for that user name.

JPI$_MAXJOBS
When you specify JPI$_MAXJOBS, $GETJPI returns the maximum number of active processes allowed for the user who owns the process specified in the call to $GETJPI. This limit is set in the UAF record of the user. The number

is returned as a word decimal value. A value of 0 means that there is no limit on the number of active processes for that user name.

JPI$_MEM

When you specify JPI$_MEM, $GETJPI returns the member number of the process's UIC, which is a longword integer value.

JPI$_MODE

When you specify JPI$_MODE, $GETJPI returns the mode of the process, which is a longword integer value. The symbolic name and value for each mode are listed in the following table; the $JPIDEF macro defines the symbolic names.

Mode Name	Value
JPI$K_OTHER	0
JPI$K_NETWORK	1
JPI$K_BATCH	2
JPI$K_INTERACTIVE	3

JPI$_MSGMASK

When you specify JPI$_MSGMASK, $GETJPI returns the default message mask of the process, which is a longword bit mask.

JPI$_OWNER

When you specify JPI$_OWNER, $GETJPI returns the process identification (PID) of the process that created the specified process. The PID is a longword hexadecimal value.

JPI$_PAGEFLTS

When you specify JPI$_PAGEFLTS, $GETJPI returns the total number of page faults incurred by the process. This is a longword integer value.

JPI$_PAGFILCNT

When you specify JPI$_PAGFILCNT, $GETJPI returns the remaining paging file quota of the process, which is a longword integer value.

JPI$_PAGFILLOC

When you specify JPI$_PAGFILLOC, $GETJPI returns the current paging file assignment of the process. The fourth byte of the returned longword value is the index of the system page file to which the process is currently assigned.

JPI$_PGFLQUOTA

When you specify JPI$_PGFLQUOTA, $GETJPI returns the paging file quota of the process, which is a longword integer value.

JPI$_PHDFLAGS

When you specify JPI$_PHDFLAGS, $GETJPI returns the process header flags as a longword bit vector.

JPI$_PID

When you specify JPI$_PID, $GETJPI returns the process identification (PID) of the process. The PID is a longword hexadecimal value.

JPI$_PPGCNT
When you specify JPI$_PPGCNT, $GETJPI returns the number of pages the process has in the working set. This is a longword integer value.

JPI$_PRCCNT
When you specify JPI$_PRCCNT, $GETJPI returns, as a longword integer value, the number of subprocesses created by the process. The number returned by JPI$_PRCCNT does not include any subprocesses created by subprocesses of the process named in the **procnam** argument.

JPI$_PRCLM
When you specify JPI$_PRCLM, $GETJPI returns the subprocess quota of the process, which is a longword integer value.

JPI$_PRCNAM
When you specify JPI$_PRCNAM, $GETJPI returns, as a character string, the name of the process. Because the process name can include up to 15 characters, the buffer length field of the item descriptor should specify 15 (bytes).

JPI$_PRI
When you specify JPI$_PRI, $GETJPI returns the current priority of the process, which is a longword integer value.

JPI$_PRIB
When you specify JPI$_PRIB, $GETJPI returns the base priority of the process, which is a longword integer value.

JPI$_PROC_INDEX
When you specify JPI$_PROC_INDEX, $GETJPI returns, as a longword integer value, the process index number of the process. The process index number is a number between 1 and the SYSGEN parameter MAXPROCESSCNT, which identifies the process. Although process index numbers are reassigned to different processes over time, at any one instant, each process in the system has a unique process index number.

You can use the process index number as an index into system global sections. Because the process index number is unique for each process, the use of it as an index into system global sections guarantees no collisions with other system processes accessing those sections.

The process index is intended to serve users who formerly used the low-order word of the PID as an index number. Because, as of Version 4.0, the meaning of the PID changed due to the introduction of clusters, the use of the low-order word of the PID as a unique identifier is no longer valid.

JPI$_PROCPRIV
When you specify JPI$_PROCPRIV, $GETJPI returns the default privileges of the process in a quadword bit mask.

JPI$_SHRFILLM
When you specify JPI$_SHRFILLM, $GETJPI returns the maximum number of open shared files allowed for the job to which the process specified in the call to $GETJPI belongs. This limit is set in the UAF record of the user who owns the process. The number is returned as a word decimal value. A value

of 0 means that there is no limit on the number of open shared files for that job.

JPI$_SITESPEC

When you specify JPI$_SITESPEC, $GETJPI returns the per-process, site-specific longword, which is a longword integer value.

JPI$_STATE

When you specify JPI$_STATE, $GETJPI returns the state of the process, which is a longword integer value. Each state has a symbolic representation. If the process is currently executing, its state is always SCH$K_CUR. The $STATEDEF macro defines the following symbols, which identify the various possible states.

State	Description
SCH$C_CEF	Common event flag wait
SCH$C_COM	Computable
SCH$C_COMO	Computable, out of balance set
SCH$C_CUR	Current process
SCH$C_COLPG	Collided page wait
SCH$C_FPG	Free page wait
SCH$C_HIB	Hibernate wait
SCH$C_HIBO	Hibernate wait, out of balance set
SCH$C_LEF	Local event flag wait
SCH$C_LEFO	Local event flag wait, out of balance set
SCH$C_MWAIT	Mutex and miscellaneous resource wait
SCH$C_PFW	Page fault wait
SCH$C_SUSP	Suspended
SCH$C_SUSPO	Suspended, out of balance set

JPI$_STS

When you specify JPI$_STS, $GETJPI returns the status flags of the process, which are contained in a longword bit vector. The $PCBDEF macro defines the following symbols for these flags:

Flag	Description
PCB$V_ASTPEN	AST pending
PCB$V_BATCH	Process is a batch job
PCB$V_DELPEN	Delete pending
PCB$V_DISAWS	Disable automatic working set adjustment
PCB$V_FORCPEN	Force exit pending
PCB$V_HARDAFF	Process bound to a particular CPU
PCB$V_HIBER	Hibernate after initial image activate
PCB$V_INQUAN	Initial quantum in progress

Flag	Description
PCB$V_INTER	Process is an interactive job
PCB$V_LOGIN	Log in without reading authorization file
PCB$V_NETWRK	Process is a network connect object
PCB$V_NOACNT	No accounting for process
PCB$V_NODELET	No delete
PCB$V_PHDRES	Process header resident
PCB$V_PREEMPTED	Kernel-mode suspend has overidden supervisor-mode suspend
PCB$V_PSWAPM	Process swap mode (1=noswap)
PCB$V_PWRAST	Power fail AST
PCB$V_RECOVER	Process can recover locks
PCB$V_RES	Resident, in balance set
PCB$V_RESPEN	Resume pending, skip suspend
PCB$V_SECAUDIT	Mandatory security auditing
PCB$V_SOFTUSP	Process is in supervisor-mode suspend
PCB$V_SSFEXC	System service exception enable (kernel)
PCB$V_SSFEXCE	System service exception enable (exec)
PCB$V_SSFEXCS	System service exception enable (super)
PCB$V_SSFEXCU	System service exception enable (user)
PCB$V_SSRWAIT	System service resource wait disable
PCB$V_SUSPEN	Suspend pending
PCB$V_SWPVBN	Write for swap VBN in progress
PCB$V_WAKEPEN	Wake pending, skip hibernate
PCB$V_WALL	Wait for all events in mask

JPI$_SWPFILLOC

When you specify JPI$_SWPFILLOC, $GETJPI returns the location of the process's swapping file, which is a longword hexadecimal value. If the number returned is positive, the fourth byte of this value identifies a specific swapping file, and the lower three bytes contain the VBN within the swapping file. If the number returned is zero or negative, the swap file location information is not currently available for the process.

JPI$_TABLENAME

When you specify JPI$_TABLENAME, $GETJPI returns the file specification of the process's current command language interpreter (CLI) table. Because the file specification can include up to 255 characters, the buffer length field in the item descriptor should specify 255 (bytes).

JPI$_TERMINAL

When you specify JPI$_TERMINAL, $GETJPI returns, for interactive users, the process's login terminal name as a character string. Because the terminal name can include up to 7 characters, the buffer length field in the item descriptor should specify 7 (bytes).

JPI$_TMBU
When you specify JPI$_TMBU, $GETJPI returns the termination mailbox unit number, which is a longword integer value.

JPI$_TQCNT
When you specify JPI$_TQCNT, $GETJPI returns the remaining timer queue entry quota of the process, which is a longword integer value.

JPI$_TQLM
When you specify JPI$_TQLM, $GETJPI returns the process's limit on timer queue entries, which is a longword integer value.

JPI$_UAF_FLAGS
When you specify JPI$_UAF_FLAGS, $GETJPI returns the UAF flags from the UAF record of the user who owns the process. The flags are returned as a longword bit vector. For a list of the symbolic names of these flags, see the UAI$_FLAGS item code under the $GETUAI system serice.

JPI$_UIC
When you specify JPI$_UIC, $GETJPI returns the UIC of the process in the standard longword format.

JPI$_USERNAME
When you specify JPI$_USERNAME, $GETJPI returns the user name of the process as a 12-byte string. If the name is less than 12 bytes, $GETJPI fills out the 12 bytes with trailing blanks.

JPI$_VIRTPEAK
When you specify JPI$_VIRTPEAK, $GETJPI returns the peak virtual address size of the process as a longword integer value.

JPI$_VOLUMES
When you specify JPI$_VOLUMES, $GETJPI returns the number of volumes that the process currently has mounted, which is a longword integer value.

JPI$_WSAUTH
When you specify JPI$_WSAUTH, $GETJPI returns the maximum authorized working set size of the process as a longword integer value.

JPI$_WSAUTHEXT
When you specify JPI$_WSAUTHEXT, $GETJPI returns the maximum authorized working set extent of the process as a longword integer value.

JPI$_WSEXTENT
When you specify JPI$_WSEXTENT, $GETJPI returns the current working set extent of the process as a longword integer value.

JPI$_WSPEAK
When you specify JPI$_WSPEAK, $GETJPI returns the peak working set size of the process as a longword integer value.

JPI$_WSQUOTA
When you specify JPI$_WSQUOTA, $GETJPI returns the working set size quota of the process as a longword integer value.

JPI$_WSSIZE

When you specify JPI$_WSSIZE, $GETJPI returns the current working set size of the process as a longword integer value.

iosb

VMS usage: **io_status_block**
type: **quadword (unsigned)**
access: **write only**
mechanism: **by reference**

I/O status block that is to receive the final completion status. The **iosb** argument is the address of the quadword I/O status block.

When you specify the **iosb** argument, $GETJPI sets the quadword to zero upon request initiation. Upon request completion, a condition value is returned to the first longword; the second longword is reserved to DIGITAL.

Though this argument is optional, DIGITAL strongly recommends that you specify it, for the following reasons:

- If you are using an event flag to signal the completion of the service, you can test the I/O status block for a condition value to be sure that the event flag was not set by an event other than service completion.
- If you are using the $SYNCH service to synchronize completion of the service, the I/O status block is a required argument for $SYNCH.
- The condition value returned in R0 and the condition value returned in the I/O status block provide information about different aspects of the call to the $GETJPI service. The condition value returned in R0 gives you information about the success or failure of the service call itself; the condition value returned in the I/O status block gives you information about the success or failure of the service operation. Therefore, to accurately assess the success or failure of the call to $GETJPI, you must check the condition values returned in both R0 and the I/O status block.

For more information about system service completion, refer to the *Introduction to VMS System Services.*

astadr

VMS usage: **ast_procedure**
type: **procedure entry mask**
access: **call without stack unwinding**
mechanism: **by reference**

AST service routine to be executed when $GETJPI completes. The **astadr** argument is the address of the entry mask of this routine.

If you specify **astadr**, the AST routine executes at the same access mode as the caller of the $GETJPI service.

astprm

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

AST parameter to be passed to the AST service routine specified by the **astadr** argument. The **astprm** argument is the longword parameter.

DESCRIPTION

The calling process must have GROUP privilege to obtain information about other processes with the same group UIC number as the calling process.

The calling process must have WORLD privilege to obtain information about other processes on the system that are not in the same group as the calling process.

Getting information about another process is an asynchronous operation because the information may be contained in the other process's virtual address space, and the process may have a lower priority or may be currently swapped out of the balance set. To allow your program to overlap other functions with the time needed to schedule the other process for execution or swap it into the balance set, $GETJPI returns immediately after it has queued its information-gathering request to the other process.

You should use the **pidadr** argument instead of the **prcnam** argument when specifying a process to $GETJPI, for the following reasons:

- The **pidadr** argument may be used to identify any process in the system, whereas the **prcnam** argument can be used only to identify processes that have the same UIC group number as the caller of $GETJPI.
- $GETJPI executes faster when you use **pidadr** rather than **prcnam**. When you specify **prcnam**, $GETJPI must search a table of process names and UICs for an entry that contains the specified process name and the UIC group number of the calling process; this search is unnecessary when you use **pidadr**.

The calling process must have GROUP privilege to obtain information about any process (except itself) in the same group. The calling process must have WORLD privilege (and must use **pidadr** rather than **prcnam**) to obtain information about a process with a different UIC group number.

Table SYS–5 shows how $GETJPI operates given various values for the **prcnam** and **pidadr** arguments.

Table SYS–5 Process Identification in $GETJPI

Process Name Specified?	Process ID Specified?	Process ID Contains	Action by $GETJPI
No	No	–	The process identification of the calling process is used, and the process identification is not returned.
No	Yes	0	The process identification of the calling process is used and returned.
No	Yes	Process ID	The process identification is used and returned.
Yes	No	–	The process name is used, and the process identification is not returned.
Yes	Yes	0	The process name is used, and the process identification is returned.
Yes	Yes	Process ID	The process identification is used and returned, and the process name is ignored.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_BADPARAM	The item list contains an invalid identifier.
SS$_ACCVIO	The item list cannot be read by the caller, or the buffer length or buffer cannot be written by the caller.
SS$_IVLOGNAM	The process name string has a length of 0 or has more than 15 characters.
SS$_NOMOREPROC	In a wildcard operation, $GETJPI found no more processes.
SS$_NONEXPR	The specified process does not exist, or an invalid process identification was specified.
SS$_NOPRIV	The process does not have the privilege to obtain information about the specified process.
SS$_SUSPENDED	The specified process is suspended or in a miscellaneous wait state, and the requested information cannot be obtained.

CONDITION VALUES RETURNED IN THE I/O STATUS BLOCK

Same as those returned in R0.

EXAMPLE

The following example shows a segment of a program used to obtain the user name of every process for which the caller has the privilege to obtain information.

```
        $JPIDEF                 ; Define $GETJPI item codes
;
PID:    .LONG   -1              ; "Wild card" PID
ITEMS:  .WORD   12              ; Size of username buffer
        .WORD   JPI$_USERNAME   ; Username item code
        .ADDRESS -
                UNAME           ; Address of username buffer
        .ADDRESS -
                UNAMES          ; Address to return username size
        .LONG   0               ; End of list
UNAME:  .BLKB   12              ; Username buffer
UNAMES: .BLKL   1               ; Username size buffer
IOSB:   .BLKQ   1               ; Completion status
;
START:  .WORD   0
          .
          .
          .
LOOP:   $GETJPI_S -
                EFN=#1, -
                PIDADR=PID, -
                ITMLST=ITEMS, -
                IOSB=IOSB
        BLBS    R0,WAIT         ; If success, continue
        CMPW    R0,#SS$_NOPRIV  ; No privilege to get info on process?
        BEQL    LOOP            ; If no priv, try next process
        CMPW    R0,#SS$_SUSPENDED ; Process suspended?
        BEQL    LOOP            ; If yes, try next process
        CMPW    R0,#SS$_NOMOREPROC ; No more processes?
        BEQL    DONE            ; If yes, all done
        BSBW    ERROR           ; Else, error
;
WAIT:   $WAITFR_S -
                EFN=#1          ; Wait for information
        MOVZWL  IOSB,R0         ; Get completion status
        BSBW    ERROR           ; Check for errors
        BSBW    DISPLAY_NAME    ; Display the name
        BRB     LOOP
```

$GETJPIW Get Job/Process Information and Wait

The Get Job/Process Information and Wait service returns information about one or more processes on the system.

The $GETJPIW service completes synchronously; that is, it returns to the caller with the requested information.

For asynchronous completion, use the Get Job/Process Information ($GETJPI) service; $GETJPI returns to the caller after queuing the information request, without waiting for the information to be returned.

In all other respects, $GETJPIW is identical to $GETJPI. For all other information about the $GETJPIW service, refer to the documentation of $GETJPI.

For additional information about system service completion, refer to the Synchronize ($SYNCH) service and to the *Introduction to VMS System Services.*

FORMAT

SYS$GETJPIW *[efn] ,[pidadr] ,[prcnam] ,itmlst [,iosb] [,astadr] [,astprm]*

$GETLKI Get Lock Information

The Get Lock Information service returns information about the lock database on a VMS system.

The $GETLKI service completes asynchronously; that is, it returns to the caller after queuing the information request, without waiting for the information to be returned.

For synchronous completion, you use the Get Lock Information and Wait ($GETLKIW) service; $GETLKIW returns to the caller with the requested information. In all other respects, $GETLKIW is identical to $GETLKI.

For additional information about system service completion, refer to the Synchronize ($SYNCH) service and to the *Introduction to VMS System Services*.

The $GETLKI, $GETLKIW, $ENQ, $ENQW, and $DEQ services together provide the user interface to the VMS lock management facility. For additional information about lock management, refer to the descriptions of these other services and to the *Introduction to VMS System Services*.

FORMAT

SYS$GETLKI *[efn] ,lkidadr ,itmlst [,iosb] [,astadr] [,astprm] [,nullarg]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

efn

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

Number of the event flag to be set when $GETLKI completes. The **efn** argument is a longword containing this number; however, $GETLKI uses only the low-order byte. If you do not specify **efn**, $GETLKI sets event flag 0.

lkidadr

VMS usage: **lock_id**
type: **longword (unsigned)**
access: **modify**
mechanism: **by reference**

Lock identification (lock ID) for the lock about which information is to be returned. The lock ID is the second longword in the lock status block, which

was created when the lock was granted. The **lkidadr** argument is the address of this longword.

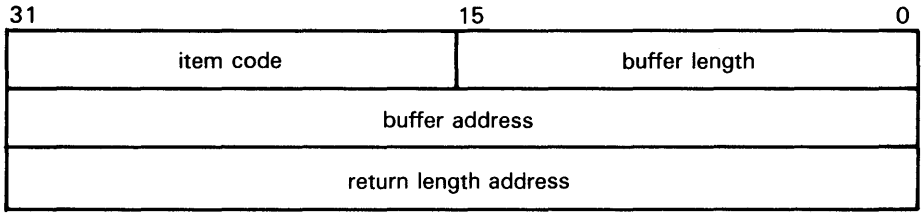
If the value specified by **lkidadr** is 0 or –1, $GETLKI assumes a wildcard operation and returns information about each lock to which the calling process has access, one lock per call.

To use the $GETLKI service, you must have read/write access to the lock ID.

itmlst

VMS usage: **item_list_3**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Item list specifying the lock information that $GETLKI is to return. The **itmlst** argument is the address of a list of item descriptors, each of which describes an item of information. The list of item descriptors is terminated by a longword of 0. The following diagram depicts the format of a single item descriptor.



ZK-1705-84

$GETLKI Item Descriptor Fields

buffer length
A word containing a user-supplied integer specifying the length (in bytes) of the buffer in which $GETLKI is to write the information. The length of the buffer needed depends upon the item code specified in the **item code** field of the item descriptor. If the value of **buffer length** is too small, $GETLKI truncates the data and returns the success condition value SS$_NORMAL.

item code
A word containing a user-supplied symbolic code specifying the item of information that $GETLKI is to return. The $LKIDEF macro defines these codes. Each item code is described in the list of $GETLKI Item Codes that follows this list of descriptor fields.

buffer address
A longword containing the user-supplied address of the buffer in which $GETLKI is to write the information.

return length address
A longword containing the user-supplied address of a longword in which $GETLKI writes return length information. This longword contains the following three bit fields.

Bits	Description
0 to 15	In this field $GETLKI writes the length in bytes of the data actually written to the buffer specified by the **buffer address** field in the item descriptor.
16 to 30	$GETLKI uses this field only when the **item code** field of the item descriptor specifies LKI$_BLOCKEDBY, LKI$_BLOCKING, or LKI$_LOCKS, each of which requests information about a list of locks. $GETLKI writes in this field the length in bytes of the information returned for a single lock in the list. You can divide this length into the total length returned for all locks (bits 0 to 15) to determine the number of locks located by that item code request.
31	$GETLKI sets this bit if the user-supplied **buffer length** argument specifies too small a buffer to contain the information returned. Note that in such a case $GETLKI will return the SS$_NORMAL condition value in R0. Therefore, to locate any faulty item descriptor, you need to check the state of bit 31 in the longword specified by the **return length** field of each item descriptor.

$GETLKI Item Codes

LKI$_BLOCKEDBY

When you specify LKI$_BLOCKEDBY, $GETLKI returns information about all locks that are currently blocked by the lock specified by **lkidadr**. The $GETLKI service returns eight items of information about each blocked lock.

The $LKIDEF macro defines the following symbolic names that refer to the eight items in the buffer.

Symbolic Name	Description
LKI$L_MSTLKID	Lock ID of the blocked lock on the system maintaining the resource (4 bytes)
LKI$L_PID	Process ID (PID) of the process that took out the blocked lock (4 bytes)
LKI$L_MSTCSID	Cluster system identifier (CSID) of the VAX node maintaining the resource that is locked by the blocked lock (4 bytes)
LKI$B_RQMODE	Lock mode requested for the blocked lock; this lock mode was specified by the **lkmode** argument in the call to $ENQ (1 byte)
LKI$B_GRMODE	Lock mode granted to the blocked lock; this lock mode is written to the lock value block (1 byte)
LKI$B_QUEUE	Name of the queue on which the blocked lock currently resides (1 byte)
LKI$L_LKID	Lock ID of the lock on the system where the lock was requested (4 bytes)
LKI$L_CSID	Cluster system identifier (CSID) of the system where the lock was requested (4 bytes)

The values that $GETLKI may write into the LKIB_RQMODE, LKIB_GRMODE, and LKI$B_QUEUE items have symbolic names; these

symbolic names specify the six lock modes and the three types of queue in which a lock may reside. The DESCRIPTION section describes these names.

Thus, the buffer specified by the **buffer address** field in the item descriptor will contain the eight items of information, repeated in sequence, for each blocked lock.

The length of the information returned for each blocked lock is returned in bits 16 to 30 of the longword specified by the **return length address** field in the item descriptor, while the total length of information returned for all blocked locks is returned in bits 0 to 15. Therefore, to determine the number of blocked locks, you divide the value of bits 16 to 30 into the value of bits 0 to 15.

LKI$_BLOCKING

When you specify LKI$_BLOCKING, $GETLKI returns information about all locks that are currently blocking the lock specified by **lkidadr**. The $GETLKI service returns eight items of information about each blocking lock.

The $LKIDEF macro defines the following symbolic names that refer to the eight items in the buffer.

Symbolic Name	Description
LKI$L_MSTLKID	Lock ID of the blocked lock on the system maintaining the resource (4 bytes)
LKI$L_PID	Process ID (PID) of the process that took out the blocking lock (4 bytes)
LKI$L_MSTCSID	Cluster system identifier (CSID) of the VAX node maintaining the resource that is locked by the blocking lock (4 bytes)
LKI$B_RQMODE	Lock mode requested for the blocking lock; this lock mode was specified by the **lkmode** argument in the call to $ENQ (1 byte)
LKI$B_GRMODE	Lock mode granted to the blocking lock; this lock mode is written to the lock value block (1 byte)
LKI$B_QUEUE	Name of the queue on which the blocking lock currently resides (1 byte)
LKI$L_LKID	Lock ID of the lock on the system where the lock was requested (4 bytes)
LKI$L_CSID	Cluster system identifier (CSID) of the system where the lock was requested (4 bytes)

The values that $GETLKI may write into the LKIB_RQMODE, LKIB_GRMODE, and LKI$B_QUEUE items have symbolic names; these symbolic names specify the six lock modes and the three types of queue in which a lock may reside. The DESCRIPTION section describes these names.

Thus, the buffer specified by the **buffer address** field in the item descriptor will contain the eight items of information, repeated in sequence, for each blocking lock.

The length of the information returned for each blocking lock is returned in bits 16 to 30 of the longword specified by the **return length address** field in the item descriptor, while the total length of information returned for all blocking locks is returned in bits 0 to 15. Therefore, to determine the number of blocking locks, you divide the value of bits 16 to 30 into the value of bits 0 to 15.

LKI$_CSID

When you specify LKI$_CSID, $GETLKI returns the Cluster System ID (CSID) of the system where the process owning the lock resides. LKI$_CSID returns the CSID of the node where the $GETLKI system service is issued when the resource is mastered on that node. When the processor is not part of a VAXcluster, LKI$_CSID returns zero.

The **buffer length** field in the item descriptor should specify 4 (bytes).

LKI$_CVTCOUNT

When you specify LKI$_CVTCOUNT, $GETLKI returns the total number of locks that are currently on the conversion queue of the resource associated with the lock. These locks are granted at one mode and are waiting to be converted to another.

The **buffer length** field in the item descriptor should specify 4 (bytes).

LKI$_GRANTCOUNT

When you specify LKI$_GRANTCOUNT, $GETLKI returns the total number of locks that are currently on the grant queue of the resource associated with the lock. Note that the total number of granted locks on the resource is equal to the sum of LKI$_CVTCOUNT and LKI$_GRANTCOUNT.

The **buffer length** field in the item descriptor should specify 4 (bytes).

Note: This item code supersedes LKI$_LCKCOUNT, which is supported in this release for compatibility with VAX/VMS Version 4.n. DIGITAL recommends that you use LKI$_GRANTCOUNT. You should update old programs with the new item code, as convenient.

LKI$_LCKREFCNT

When you specify LKI$_LCKREFCNT, $GETLKI returns the number of locks that have this lock as a parent lock. When these locks were created, the **parid** argument in the call to $ENQ or $ENQW specified the lock ID of this lock.

The **buffer length** field in the item descriptor should specify 4 (bytes).

LKI$_LKID

When you specify LKI$_LKID, $GETLKI returns the lock ID of the lock on the system where the process owning the lock resides. The lock ID returned by this item code is meaningful only on the system specified in the value returned by the LKI$_CSID item code.

The **buffer length** field in the item descriptor should specify 4 (bytes).

Note: This item code and LKI$_MSTLKID supersede LKI$_REMLKID, which is supported for compatibility with VAX/VMS Version 4.n. DIGITAL recommends that you use the new item codes. You should update old programs with the new item codes, as convenient.

LKI$_LOCKID

When you specify LKI$_LOCKID, $GETLKI returns the lock ID of the current lock. The current lock is the one specified by the **lkidadr** argument unless **lkidadr** is specified as -1 or 0, which indicates a wildcard operation. Thus, this item code is usually specified only in wildcard operations where it is useful to know the lock IDs of the locks that $GETLKI has discovered in the wildcard operation.

The lock ID is a longword value, so the **buffer length** field in the item descriptor should specify 4 (bytes).

LKI$_LOCKS

When you specify LKI$_LOCKS, $GETLKI returns information about all locks on the resource associated with the lock specified by **lkidadr**. These locks are the sum of blocking locks and blocked locks.

The $LKIDEF macro defines the following symbolic names that refer to the eight items in the buffer.

Symbolic Name	Description
LKI$L_MSTLKID	Lock ID of the blocked lock on the system maintaining the resource (4 bytes)
LKI$L_PID	Process ID (PID) of the process that took out the lock (4 bytes)
LKI$L_MSTCSID	Cluster system identifier (CSID) of the VAX node maintaining the resource that is locked by the lock (4 bytes)
LKI$B_RQMODE	Lock mode requested for the lock; this lock mode was specified by the **lkmode** argument in the call to $ENQ (1 byte)
LKI$B_GRMODE	Lock mode granted to the lock; this lock mode is written to the lock value block (1 byte)
LKI$B_QUEUE	Name of the queue on which the lock currently resides (1 byte)
LKI$L_LKID	Lock ID of the lock on the system where the lock was requested (4 bytes)
LKI$L_CSID	Cluster system identifier (CSID) of the system where the lock was requested (4 bytes)

The values that $GETLKI may write into the LKIB_RQMODE, LKIB_GRMODE, and LKI$B_QUEUE items have symbolic names; these symbolic names specify the six lock modes and the three types of queue in which a lock may reside. The DESCRIPTION section describes these names.

Thus, the buffer specified by the **buffer address** field in the item descriptor will contain the eight items of information, repeated in sequence, for each lock.

The length of the information returned for each lock is returned in bits 16 to 30 of the longword specified by the **return length address** field in the item descriptor, while the total length of information returned for all locks is returned in bits 0 to 15. Therefore, to determine the number of locks, you divide the value of bits 16 to 30 into the value of bits 0 to 15.

LKI$_MSTCSID

When you specify LKI$_MSTCSID, $GETLKI returns the Cluster System ID (CSID) of the node currently mastering the resource that is associated with the specified lock. Although the resource may be locked by processes on any node in the cluster, the resource itself is maintained on a single node. You can use the DCL command SHOW CLUSTER or the $GETSYI service to determine which VAX node in the cluster is identified by the CSID that $GETLKI returns.

Because the processor mastering the lock can change at any time, multiple calls to $GETLKI for the same lock may produce different values for this item code. LKI$_MSTCSID returns the CSID of the node where the $GETLKI system service is issued when the resource is mastered on that node. When the processor where the $GETLKI was issued is not part of a VAXcluster, this item code returns zero.

The **buffer length** field in the item descriptor should specify 4 (bytes).

Note: This item code supersedes LKI$_SYSTEM, which is supported for compatibility with VAX/VMS Version 4.n. DIGITAL recommends that you use LKI$_MSTCSID. You should update old programs with the new item code, as convenient.

LKI$_MSTLKID

When you specify LKI$_MSTLKID, $GETLKI returns the lock ID for the current master copy of the lock. Although the resource may be locked by processes on any node in the cluster, the resource itself is maintained on a single node. Because lock IDs are unique to each processor on a VAXcluster, the lock ID returned by this item code has meaning only for the processor that is specified in the value returned by the LKI$_MSTCSID item code.

Because the processor mastering the lock can change at any time, multiple calls to $GETLKI for the same lock may produce different values for this item code. When the lock is mastered on the node where the $GETLKI system service is issued, or when the node is not a member of a VAXcluster, this item code returns the same information as LKI$_LKID.

The **buffer length** field in the item descriptor should specify 4 (bytes).

Note: This item code and LKI$_LKID supersede LKI$_REMLKID, which is supported for compatibility with VAX/VMS Version 4.n. DIGITAL recommends that you use LKI$_MSTLKID and LKI$_LKID. You should update old programs with the new item codes, as convenient.

LKI$_NAMSPACE

When you specify LKI$_NAMSPACE, $GETLKI returns information about the resource name space. This information is contained in a longword consisting of four bit fields; therefore, the **buffer length** field in the item descriptor should specify 4 (bytes).

Each of the four bit fields may be referred to by their symbolic names; the $LKIDEF macro defines the symbolic names. The following table lists, in order, the symbolic name of each bit field:

Symbolic Name	Description
LKI$W_GROUP	In this field (bits 0 to 15) $GETLKI writes the UIC group number of the process that took out the first lock on the resource, thereby creating the resource name. This process issued a call to $ENQ or $ENQW specifying the name of the resource in the **resnam** argument. However, if this process specified the LCK$_SYSTEM flag in the call to $ENQ or $ENQW, the resource name is systemwide. In this case, the UIC group number of the process is not associated with the resource name. Consequently, this field (bits 0 to 15) is significant only if the resource name is not systemwide. $GETLKI sets bit 31 if the resource name is systemwide.
LKI$B_RMOD	In this field (bits 16 to 23) $GETLKI writes the access mode associated with the first lock taken out on the resource.
LKI$B_STATUS	This field (bits 24 to 30) is not used. $GETLKI sets it to 0.
LKI$V_SYSNAM	This field (bit 31) indicates whether the resource name is systemwide. $GETLKI sets this bit if the resource name is systemwide and clears it if the resource name is qualified by the creating process's UIC group number. The state of this bit determines the interpretation of bits 0 to 15.

LKI$_PARENT

When you specify LKI$_PARENT, $GETLKI returns the lock ID of the parent lock for the lock, if a parent lock was specified in the call to $ENQ or $ENQW. If the lock does not have a parent lock, $GETLKI returns the value 0.

Because the parent lock ID is a longword, the **buffer length** field in the item descriptor should specify 4 (bytes).

LKI$_PID

When you specify LKI$_PID, $GETLKI returns the process identification (process ID) of the process that owns the lock.

The process ID is a longword value, so the **buffer length** field in the item descriptor should specify 4 (bytes).

LKI$_RESNAM

When you specify LKI$_RESNAM, $GETLKI returns the resource name string and its length, which must be from 1 to 31 bytes. The resource name string was specified in the **resnam** argument in the initial call to $ENQ or $ENQW.

The $GETLKI service returns the length of the string in the **return length address** field in the item descriptor. However, in the call to $GETLKI, you do not know how long the string is. Therefore, to avoid buffer overflow, you should specify the maximum length (31 bytes) in the **buffer length** field in the item descriptor.

LKI$_RSBREFCNT

When you specify LKI$_RSBREFCNT, $GETLKI returns the number of subresources of the resource associated with the lock. A subresource has the resource as a parent resource. Note, however, that the number of subresources may differ from the number of sublocks of the lock, because

any number of processes may lock the resource. If any of these processes then locks another resource, and in doing so specifies the lock ID of the lock on the first resource as a parent lock, then the second resource becomes a subresource of the first resource.

Thus, the number of sublocks on a lock is limited to the number of sublocks that a single process takes out, whereas the number of subresources on a resource is determined by (potentially) multiple processes.

The subresource reference count is a longword value, so the **buffer length** field in the item descriptor should specify 4 (bytes).

LKI$_STATE

When you specify LKI$_STATE, $GETLKI returns the current state of the lock. The current state of the lock is described by the following three 1-byte items (in the order specified): (1) the lock mode requested (in the call to $ENQ or $ENQW) for the lock, (2) the lock mode granted (by $ENQ or $ENQW) for the lock, and (3) the name of the queue on which the lock currently resides.

The **buffer length** field in the item descriptor should specify 3 (bytes). The $LKIDEF macro defines the following symbolic names that refer to the three 1-byte items in the buffer.

Symbolic Name	Description
LKI$B_STATE_RQMODE	Lock mode requested
LKI$B_STATE_GRMODE	Lock mode granted
LKI$B_STATE_QUEUE	Name of queue on which the lock resides

The values that $GETLKI may write into each 1-byte item have symbolic names; these symbolic names specify the six lock modes and the three types of queue in which a lock may reside. The DESCRIPTION section describes these names.

LKI$_VALBLK

When you specify LKI$_VALBLK, $GETLKI returns the lock value block of the locked resource. This lock value block is the master copy that the lock manager maintains for the resource, not the process-private copy.

Because the lock value block is 16 bytes, the **buffer length** field in the item descriptor should specify 16.

LKI$_WAITCOUNT

When you specify LKI$_WAITCOUNT, $GETLKI returns the total number of locks that are currently on the wait queue of the resource associated with the lock. These locks are waiting to be granted.

The **buffer length** field in the item descriptor should specify 4 (bytes).

iosb

VMS usage: **io_status_block**
type: **quadword (unsigned)**
access: **write only**
mechanism: **by reference**

I/O Status Block that is to receive the final completion status. The **iosb** argument is the address of a quadword.

When $GETLKI is called, it sets the I/O status block to 0. When $GETLKI completes, it writes a condition value to the first longword in the quadword. The remaining two words in the quadword are unused.

Although this argument is optional, DIGITAL strongly recommends that you specify it, for the following reasons:

- If you are using an event flag to signal the completion of the service, you can test the I/O status block for a condition value to be sure that the event flag was not set by an event other than service completion.
- If you are using the $SYNCH service to synchronize completion of the service, the I/O status block is a required argument for $SYNCH.
- The condition value returned in R0 and the condition value returned in the I/O status block provide information about different aspects of the call to the $GETLKI service. The condition value returned in R0 gives you information about the success or failure of the service call itself; the condition value returned in the I/O status block gives you information about the success or failure of the service operation. Therefore, to accurately assess the success or failure of the call to $GETLKI, you must check the condition values returned in both R0 and the I/O status block.

astadr

VMS usage: **ast_procedure**
type: **procedure entry mask**
access: **call without stack unwinding**
mechanism: **by reference**

AST service routine to be executed when the service completes. The **astadr** argument is the address of the entry mask of this routine.

If you specify this argument, the AST routine executes at the same access mode as the caller of the $GETLKI service.

astprm

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

AST parameter to be passed to the AST service routine specified by the **astadr** argument. The **astprm** argument is the longword parameter.

nullarg

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

Place-holding argument reserved by DIGITAL.

DESCRIPTION

Depending on the operation, the calling process may need a certain privilege to use $GETLKI:

- You need WORLD privilege to obtain information about locks held by processes in other groups.
- To obtain information about system locks, you need either SYSLCK privilege, or the process must be executing in executive or kernel access mode.

The access mode of the calling process must be equal to or more privileged than the access mode at which the lock was initially granted.

When locking on a resource is clusterwide, a single master copy of the resource is maintained on the node that owns the process that created the resource by taking out the first lock on it. When a process on another VAX node locks that same resource, a local copy of the resource is copied to the node and the lock is identified by a lock ID that is unique to that node.

In a VAXcluster environment, however, you cannot use $GETLKI to obtain directly information about locks on other nodes in the cluster; that is, you cannot specify in a call to $GETLKI the lock ID of a lock held by a process on another node. The $GETLKI service interprets the **lkidadr** argument as the lock ID of a lock on the caller's node, even though the resource associated with a lock may or may not have its master copy on the caller's node.

However, because a process on another node in the cluster may have a lock on the same resource as the caller of $GETLKI, the caller, in obtaining information about the resource, may indirectly obtain some information about locks on the resource that are held by processes on other nodes. One example of information indirectly obtained about a resource is the contents of lock queues; these queues contain information about all locks on the resource, and some of these locks may be held by processes on other nodes.

Another example of information more directly obtained is the remote lock ID of a lock held by a process on another node. Specifically, if the caller of $GETLKI on node A specifies a lock (by means of **lkidadr**) and that lock is held by a process on node B, $GETLKI will return the lock ID of the lock from node B's lock database if the LKI$_REMLKID item code is specified in the call.

Item codes LKI$_BLOCKEDBY, LKI$_BLOCKING, LKI$_LOCKS, and LKI$_STATE specify that $GETLKI return various items of information; some of these items are the names of lock modes or the names of lock queues. The $LCKDEF macro defines the following symbolic names:

Symbolic Name	Lock Mode
LCK$K_NLMODE	Null mode
LCK$K_CRMODE	Concurrent read mode
LCK$K_CWMODE	Concurrent write mode
LCK$K_PRMODE	Protected read mode
LCK$K_PWMODE	Protected write mode
LCK$K_EXMODE	Exclusive mode

Symbolic Name	Queue Name
LKI$C_GRANTED	Granted queue, holding locks that have been granted
LKI$C_CONVERT	Converting queue, holding locks that are currently being converted to another lock mode
LKI$C_WAITING	Waiting queue, holding locks that are neither granted nor converting (for example, a blocked lock)

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_NOMORELOCK	The caller requested a wildcard operation by specifying a value of 0 or -1 for the **lkidadr** argument, and $GETLKI has exhausted the locks about which it can return information to the caller; or no **lkidadr** argument is specified. This is an alternate success status.
SS$_ACCVIO	The item list cannot be read; the areas specified by the **buffer address** and **return length address** fields in the item descriptor cannot be written; or the location specified by the **lkidadr** argument cannot be written.
SS$_BADPARAM	You specified an invalid item code.
SS$_IVLOCKID	The **lkidadr** argument specified an invalid lock ID.
SS$_IVMODE	A more privileged access mode is required.
SS$_NOSYSLCK	The caller attempted to acquire information about a systemwide lock and did not have the required SYSLCK privilege.
SS$_NOWORLD	The caller attempted to acquire information about a lock held by a process in another group and did not have the required WORLD privilege.

SS$_EXQUOTA	The caller has insufficient ASTLM or BYTLM quota.
SS$_INSFMEM	The nonpaged dynamic memory is insufficient for the operation.

CONDITION VALUES RETURNED IN THE I/O STATUS BLOCK

Same as those returned in R0.

$GETLKIW Get Lock Information and Wait

The Get Lock Information and Wait service returns information about the lock database on a VMS system.

The $GETLKIW service completes synchronously; that is, it returns to the caller with the requested information.

For asynchronous completion, you use the Get Lock Information ($GETLKI) service; $GETLKI returns to the caller after queueing the information request, without waiting for the information to be returned.

In all other respects, $GETLKIW is identical to $GETLKI. For all other information about the $GETLKIW service, refer to the documentation of $GETLKI.

For additional information about system service completion, refer to the Synchronize ($SYNCH) service and to the *Introduction to VMS System Services*.

The $GETLKI, $GETLKIW, $ENQ, $ENQW, and $DEQ services together provide the user interface to the VMS lock management facility. Refer to the descriptions of these other services and to the *Introduction to VMS System Services* for additional information about lock management.

FORMAT

SYS$GETLKIW *[efn] ,lkidadr ,itmlst [,iosb] [,astadr] [,astprm] [,nullarg]*

$GETMSG Get Message

The Get Message service locates and returns message text associated with a given message identification code into the caller's buffer. The message can be from the system message file or a user-defined message.

FORMAT

SYS$GETMSG *msgid ,msglen ,bufadr ,[flags] ,[outadr]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

msgid

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

Identification of the ssage to be retrieved. The **msgid** argument is a longword value cont ig the message identification. Each message has a unique identification, contained in bits 3 through 27 of system longword condition values.

msglen

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

Length of the message string returned by $GETMSG. The **msglen** argument is the address of a word into which $GETMSG writes this length.

bufadr

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

Buffer to receive the message string. The **bufadr** argument is the address of a character string descriptor pointing to the buffer into which $GETMSG writes the message string. The maximum size of any message string is 256 bytes.

flags

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

Message components to be returned. The **flags** argument is a longword bit vector wherein a bit, when set, specifies that the message component is to be returned. The following table describes the significant bits:

Bit	Value	Description
0	1	Include text of message
	0	Do not include text of message
1	1	Include message identifier
	0	Do not include message identifier
2	1	Include severity indicator
	0	Do not include severity indicator
3	1	Include facility name
	0	Do not include facility name

If you omit this argument in a VAX MACRO or BLISS-32 service call, it defaults to a value of 15; that is, all flags are set and all components of the message are returned. If you omit this argument in a FORTRAN service call, it defaults to a value of 0; the value 0 causes $GETMSG to use the process default flags.

outadr

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

Optional information to be returned by $GETMSG. The **outadr** argument is the address of a 4-byte array into which $GETMSG writes the following information:

Byte	Contents
0	Reserved
1	Count of FAO arguments associated with message
2	User-specified value in message, if any
3	Reserved

DESCRIPTION

VMS uses this service to retrieve messages based on unique message identifications and to prepare to output the messages.

The message identifications correspond to the symbolic names for condition values returned by system components, for example, SS$_code from system services, RMS$_code for VMS RMS messages, and so on.

When you set all bits in the **flags** argument, $GETMSG returns a string in the following format:

facility-severity-ident, message-text

where:

facility	Identifies the component of the operating system.
severity	Is the severity code (the low-order 3 bits of the condition value).
ident	Is the unique message identifier.
message-text	Is the text of the message.

For example, if you specify the MSGID=#SS$_DUPLNAM argument, the $GETMSG service returns the following string:

```
%SYSTEM-F-DUPLNAM, duplicate process name
```

You can define your own messages with the Message Utility. See the *VMS Message Utility Manual* for additional information.

The message text associated with a particular 32-bit message identification can be retrieved from one of several places. This service takes the following steps to locate the message text:

1. All message sections linked into the currently executing image are searched for the associated information.
2. If the information is not found, the process-permanent message file is searched. (You can specify the process-permanent message file by using the SET MESSAGE command.)
3. If the information is not found, the system-wide message file is searched.
4. If the information is not found, the SS$_MSGNOTFND condition value is returned in R0 and a message in the following form is returned to the caller's buffer:

```
%facility-severity-NONAME, message=xxxxxxxx[hex],
     (facility=n, message=n[dec])
```

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_BUFFEROVF	The service completed successfully. The string returned overflowed the buffer provided and has been truncated.
SS$_INSFARG	The call arguments are insufficient.
SS$_MSGNOTFND	The service completed successfully; however, the message code cannot be found, and a default message has been returned.

EXAMPLE

The following example shows a segment of a program used to obtain only the text portion of the message associated with the system message code SS$_DUPLNAM. The $GETMSG service returns the following string:

duplicate process name

```
CODE:    .LONG   SS$_DUPLNAM             ; Message identification
LENGTH:  .WORD   0
BUFDESC:
         .LONG   256
         .ADDRESS -
                 BUFFER
BUFFER:  .BLKB   256
FLAGS:   .WORD   ^B0001                  ; Message flags - text only
           .
           .
           .
         $GETMSG_S -
                 MSGID=CODE, -
                 MSGLEN=LENGTH, -
                 BUFADR=BUFDESC, -
                 FLAGS=FLAGS
```

$GETQUI Get Queue Information

The Get Queue Information service returns information about queues and the jobs initiated from those queues. The $GETQUI and $SNDJBC services together provide the user interface to the VMS Job Controller, which is the VMS queue and accounting manager. See the DESCRIPTION section of the $SNDJBC service for a discussion of the different types of jobs and queues.

The $GETQUI service completes asynchronously; that is, it returns to the caller after queuing the request, without waiting for the operation to complete.

For synchronous completion, you use the Get Queue Information and Wait ($GETQUIW) service. The $GETQUIW service is identical to $GETQUI in every way except that $GETQUIW returns to the caller after the operation has completed.

For additional information about system service completion, refer to the Synchronize ($SYNCH) service and to the *Introduction to VMS System Services*.

FORMAT

SYS$GETQUI *[efn] ,func [,nullarg] [,itmlst] [,iosb] [,astadr] [,astprm]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

efn

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

Number of the event flag to be set when $GETQUI completes. The **efn** argument is a longword containing this number; however, $GETQUI uses only the low-order byte. The **efn** argument is optional.

When the request is queued, $GETQUI clears the specified event flag (or event flag 0 if **efn** was not specified). Then, when the operation completes, $GETQUI sets the specified event flag (or event flag 0).

func

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

Function code specifying the function that $GETQUI is to perform. The **func** argument is a word containing this function code. The $QUIDEF macro defines the names of each function code.

You can specify only one function code in a single call to $GETQUI. Most function codes require or allow for additional information to be passed in the call. You pass this information by using the **itmlst** argument, which specifies a list of one or more item descriptors. Each item descriptor in turn specifies an item code, which either describes the specific information to be returned by $GETQUI, or otherwise affects the action designated by the function code.

You can use wildcard mode to make a sequence of calls to $GETQUI to get information about all characteristics, form definitions, queues, or jobs contained in the system job queue file. For information on using wildcard mode, see the DESCRIPTION section.

The following list specifies each function code, describes the action it designates, and lists which item code or codes are applicable; descriptions of the item codes appear in the description of the **itmlst** argument.

$GETQUI Function Codes with Their Valid Item Codes

QUI$_CANCEL_OPERATION

This request terminates any wildcard operation that may have been initiated by a previous call to $GETQUI by releasing the GETQUI context block (GQC) that the system maintains for your process. Because only one wildcard search sequence can be outstanding at any one time, you do not have to specify any item codes.

When you call $GETQUI to perform a series of wildcard requests to retrieve information about characteristics, forms, queues (and their associated jobs and files) or job entries, the job controller maintains a GQC between calls that points to the next object in the wildcard sequence. The system retains this information until (1) you have made calls to $GETQUI to examine every object in the sequence; (2) your process has terminated; or (3) you explicitly cancel the wildcard operation by using the QUI$_CANCEL_OPERATION function code. If your calls to $GETQUI have located all the objects in the sequence in which you are interested, you should terminate the wildcard operation. This frees job controller resources and allows you to initiate another $GETQUI operation.

QUI$_DISPLAY_CHARACTERISTIC

This request returns information about a specific characteristic definition, or the next characteristic definition in a wildcard operation.

A successful QUI$_DISPLAY_CHARACTERISTIC wildcard operation terminates when the $GETQUI service has returned information about all characteristic definitions included in the wildcard sequence. The $GETQUI service indicates termination of this sequence by returning the condition value JBC$_NOMORECHAR in the I/O status block. If the $GETQUI service does not find any characteristic definitions, it returns the condition value JBC$_NOSUCHCHAR in the I/O status block.

For more information on how to request information about characteristics, see the DESCRIPTION section.

You must specify one of the following input item codes; you may specify both:

QUI$_SEARCH_NAME
QUI$_SEARCH_NUMBER

You may specify the following input item code:

QUI$_SEARCH_FLAGS

You may specify the following output item codes:

QUI$_CHARACTERISTIC_NAME
QUI$_CHARACTERISTIC_NUMBER

QUI$_DISPLAY_ENTRY

This request returns information about a specific job entry, or the next job entry that matches the selection criteria in a wildcard operation. You use the QUI$_SEARCH_NUMBER item code to specify the job entry number.

In wildcard mode, the QUI$_DISPLAY_ENTRY operation also establishes a job context for subsequent QUI$_DISPLAY_FILE operations. The job context established remains in effect until you make another call to the $GETQUI service that specifies either the QUI$_DISPLAY_ENTRY or QUI$_CANCEL_OPERATION function code.

A successful QUI$_DISPLAY_ENTRY wildcard operation terminates when the $GETQUI service has returned information about all job entries for the specified user (or the current user name if the QUI$_SEARCH_USERNAME item code is not specified). The $GETQUI service signals termination of this sequence by returning the condition value JBC$_NOMOREENT in the I/O status block. If the $GETQUI service does not find a job with the specified entry number, or does not find a job meeting the search criteria, it returns the condition value JBC$_NOSUCHENT in the first longword of the I/O status block.

You may specify the following input item codes:

QUI$_SEARCH_USERNAME
QUI$_SEARCH_NUMBER
QUI$_SEARCH_FLAGS

You may specify the following output item codes:

QUI$_ACCOUNT_NAME
QUI$_AFTER_TIME
QUI$_CHARACTERISTICS
QUI$_CHECKPOINT_DATA
QUI$_CLI
QUI$_COMPLETED_BLOCKS
QUI$_CONDITION_VECTOR
QUI$_CPU_LIMIT
QUI$_ENTRY_NUMBER
QUI$_FORM_NAME
QUI$_FORM_STOCK
QUI$_INTERVENING_BLOCKS
QUI$_INTERVENING_JOBS

QUI$_JOB_COPIES
QUI$_JOB_COPIES_DONE
QUI$_JOB_FLAGS
QUI$_JOB_NAME
QUI$_JOB_PID
QUI$_JOB_SIZE
QUI$_JOB_STATUS
QUI$_LOG_QUEUE
QUI$_LOG_SPECIFICATION
QUI$_NOTE
QUI$_OPERATOR_REQUEST
QUI$_PARAMETER_1 through 8
QUI$_PENDING_JOB_REASON
QUI$_PRIORITY
QUI$_QUEUE_NAME
QUI$_RESTART_QUEUE_NAME
QUI$_REQUEUE_QUEUE_NAME
QUI$_SUBMISSION_TIME
QUI$_UIC
QUI$_USERNAME
QUI$_WSDEFAULT
QUI$_WSEXTENT
QUI$_WSQUOTA

QUI$_DISPLAY_FILE

This request returns information about the next file defined for the current job context. You normally make this request as part of a nested wildcard sequence of queue-job-file operations or a nested wildcard sequence of entry-file operations; that is, before you make a call to $GETQUI to request file information, you have already made a call to the $GETQUI service to establish the job context of the job that contains the files in which you are interested.

The $GETQUI service signals that it has returned information about all the files defined for the current job context by returning the condition value JBC$_NOMOREFILE in the I/O status block. If the current job context contains no files, $GETQUI returns the condition value JBC$_NOSUCHFILE in the I/O status block.

A batch job can make a call to the $GETQUI service to request information about the command file that is currently executing without first making calls to the service to establish a queue and job context. To do this, the batch job specifies the QUI$V_SEARCH_THIS_JOB option of the QUI$_SEARCH_FLAGS item code. The system does not save the queue or job context established in such a call.

For more information about how to request file information, see the DESCRIPTION section.

You may specify the following input item code:

QUI$_SEARCH_FLAGS

You may specify the following output item codes:

QUI$_FILE_COPIES
QUI$_FILE_COPIES_DONE
QUI$_FILE_FLAGS
QUI$_FILE_IDENTIFICATION

QUI$_FILE_SETUP_MODULES
QUI$_FILE_SPECIFICATION
QUI$_FILE_STATUS
QUI$_FIRST_PAGE
QUI$_LAST_PAGE

QUI$_DISPLAY_FORM

This request returns information about a specific form definition, or the next form definition in a wildcard operation.

A successful QUI$_DISPLAY_FORM wildcard operation terminates when the $GETQUI service has returned information about all form definitions included in the wildcard sequence. The $GETQUI service signals termination of this wildcard sequence by returning the condition value JBC$_NOMOREFORM in the I/O status block. If the $GETQUI service finds no form definitions, it returns the condition value JBC$_NOSUCHFORM in the I/O status block.

For more information on how to request information about forms, see the DESCRIPTION section.

You must specify one of the following input item codes. You may specify both:

QUI$_SEARCH_NAME
QUI$_SEARCH_NUMBER

You may specify the following input item code:

QUI$_SEARCH_FLAGS

You may specify the following output item codes:

QUI$_FORM_DESCRIPTION
QUI$_FORM_FLAGS
QUI$_FORM_LENGTH
QUI$_FORM_MARGIN_BOTTOM
QUI$_FORM_MARGIN_LEFT
QUI$_FORM_MARGIN_RIGHT
QUI$_FORM_MARGIN_TOP
QUI$_FORM_NAME
QUI$_FORM_NUMBER
QUI$_FORM_SETUP_MODULES
QUI$_FORM_STOCK
QUI$_FORM_WIDTH
QUI$_PAGE_SETUP_MODULES

QUI$_DISPLAY_JOB

This request returns information about the next job defined for the current queue context. You normally make this request as part of a nested wildcard queue-job sequence of operations; that is, before you make a call to $GETQUI to request job information, you have already made a call to the $GETQUI service to establish the queue context of the queue that contains the job in which you are interested.

In wildcard mode, the QUI$_DISPLAY_JOB operation also establishes a job context for subsequent QUI$_DISPLAY_FILE operations. The job context established remains in effect until another call is made to the $GETQUI service that specifies the QUI$_DISPLAY_JOB, QUI$_DISPLAY_QUEUE, or QUI$_CANCEL_OPERATION function code.

The $GETQUI service signals that it has returned information about all the jobs contained in the current queue context by returning the condition value JBC$_NOMOREJOB in the I/O status block. If the current queue context contains no jobs, $GETQUI returns the condition value JBC$_NOSUCHJOB in the first longword of the I/O status block.

A batch job can make a call to the $GETQUI service to request information about itself without first making a call to the service to establish a queue context. To do this, the batch job must specify the QUI$V_SEARCH_THIS_JOB option of the QUI$_SEARCH_FLAGS item code. The system does not save the queue or job context established in such a call.

For more information about how to request job information, see the DESCRIPTION section.

You may specify the following input item code:

QUI$_SEARCH_FLAGS

You may specify the following output item codes:

QUI$_ACCOUNT_NAME
QUI$_AFTER_TIME
QUI$_CHARACTERISTICS
QUI$_CHECKPOINT_DATA
QUI$_CLI
QUI$_COMPLETED_BLOCKS
QUI$_CONDITION_VECTOR
QUI$_CPU_LIMIT
QUI$_ENTRY_NUMBER
QUI$_FORM_NAME
QUI$_FORM_STOCK
QUI$_INTERVENING_BLOCKS
QUI$_INTERVENING_JOBS
QUI$_JOB_COPIES
QUI$_JOB_COPIES_DONE
QUI$_JOB_FLAGS
QUI$_JOB_NAME
QUI$_JOB_PID
QUI$_JOB_SIZE
QUI$_JOB_STATUS
QUI$_LOG_QUEUE
QUI$_LOG_SPECIFICATION
QUI$_NOTE
QUI$_OPERATOR_REQUEST
QUI$_PARAMETER_1 through 8
QUI$_PENDING_JOB_REASON
QUI$_PRIORITY
QUI$_QUEUE_NAME
QUI$_REQUEUE_QUEUE_NAME
QUI$_RESTART_QUEUE_NAME
QUI$_SUBMISSION_TIME
QUI$_UIC
QUI$_USERNAME
QUI$_WSDEFAULT
QUI$_WSEXTENT
QUI$_WSQUOTA

QUI$_DISPLAY_QUEUE

This request returns information about a specific queue definition, or the next queue definition in a wildcard operation.

In wildcard mode, the QUI$_DISPLAY_QUEUE operation also establishes a queue context for subsequent QUI$_DISPLAY_JOB operations. The queue context established remains in effect until another call is made to the $GETQUI service that specifies either the QUI$_DISPLAY_QUEUE or QUI$_CANCEL_OPERATION function code.

The $GETQUI service indicates that it has returned information about all the queues contained in the current wildcard sequence by returning the condition value JBC$_NOMOREQUE in the I/O status block. If no queue is found, $GETQUI returns the condition value JBC$_NOSUCHQUE in the first longword of the I/O status block.

A batch job can make a call to the $GETQUI service to request information about the queue in which it is contained without first making a call to the service to establish a queue context. To do this, the batch job must specify the QUI$V_SEARCH_THIS_JOB option of the QUI$_SEARCH_FLAGS item code. The system does not save the queue context established in such a call.

For more information about how to request queue information, see the DESCRIPTION section.

You must specify the following input item code:

QUI$_SEARCH_NAME

You may specify the following input item code:

QUI$_SEARCH_FLAGS

You may specify the following output item codes:

QUI$_ASSIGNED_QUEUE_NAME
QUI$_BASE_PRIORITY
QUI$_CHARACTERISTICS
QUI$_CPU_DEFAULT
QUI$_CPU_LIMIT
QUI$_DEFAULT_FORM_NAME
QUI$_DEFAULT_FORM_STOCK
QUI$_DEVICE_NAME
QUI$_EXECUTING_JOB_COUNT
QUI$_FORM_NAME
QUI$_FORM_STOCK
QUI$_GENERIC_TARGET
QUI$_HOLDING_JOB_COUNT
QUI$_JOB_LIMIT
QUI$_JOB_RESET_MODULES
QUI$_JOB_SIZE_MAXIMUM
QUI$_JOB_SIZE_MINIMUM
QUI$_LIBRARY_SPECIFICATION
QUI$_OWNER_UIC
QUI$_PENDING_JOB_BLOCK_COUNT
QUI$_PENDING_JOB_COUNT
QUI$_PROCESSOR
QUI$_PROTECTION
QUI$_QUEUE_DESCRIPTION
QUI$_QUEUE_FLAGS

QUI$_QUEUE_NAME
QUI$_QUEUE_STATUS
QUI$_RETAINED_JOB_COUNT
QUI$_SCSNODE_NAME
QUI$_TIMED_RELEASE_JOB_COUNT
QUI$_WSDEFAULT
QUI$_WSEXTENT
QUI$_WSQUOTA

QUI$_TRANSLATE_QUEUE

This request translates a logical name for a queue to the equivalence name for the queue. The logical name is specified by QUI$_SEARCH_NAME. The translation is performed iteratively until the equivalence string is found or the number of translations allowed by the system has been reached.

You must specify the following input item code:

QUI$_SEARCH_NAME

You may specify the following output item code:

QUI$_QUEUE_NAME

nullarg

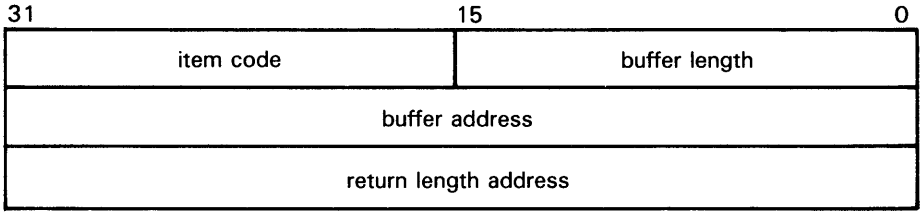
VMS usage: **null_arg**
type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

Place-holding argument reserved by DIGITAL.

itmlst

VMS usage: **item_list_3**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Item list supplying information to be used in performing the function specified by the **func** argument. The **itmlst** argument is the address of the item list. The item list consists of one or more item descriptors, each of which contains an item code. The item list is terminated by an item code of 0 or by a longword of 0. The following diagram depicts the structure of a single item descriptor.



ZK-1705-84

$GETQUI Item Descriptor Fields

buffer length

A word specifying the length of the buffer; the buffer either supplies input information for $GETQUI or receives information from $GETQUI. The required length of the buffer varies depending on the item code specified and is given in the description of each item code.

item code

A word containing an item code, which identifies the nature of the information supplied for $GETQUI or that is received from $GETQUI. Each item code has a symbolic name; the $QUIDEF macro defines these symbolic names that have the following format:

QUI$_code

There are two types of item code:

- Input value item code. The $GETQUI service has only three input value item codes: QUI$_SEARCH_NAME, QUI$_SEARCH_NUMBER, and QUI$_SEARCH_FLAGS. These item codes specify the object name or number for which $GETQUI is to return information, and the extent of $GETQUI's search for these objects. Most function codes require that you specify at least one input value item code. The function code or codes for which each item code is valid is shown in parentheses after the item code description.

  For input value item codes, the **buffer length** and **buffer address** fields of the item descriptor must be nonzero; the return length field must be zero. Specific buffer length requirements are given in the description of each item code.

- Output value item code. Output value item codes specify a buffer for information returned by $GETQUI. For output value item codes, the **buffer length** and **buffer address** fields of the item descriptor must be nonzero; the return length field may be zero or nonzero. Specific buffer length requirements are given in the description of each item code.

Several item codes specify a queue name, form name, or characteristic name to $GETQUI, or request that $GETQUI return one of these names. For these item codes, the buffer must specify or be prepared to receive a string containing from 1 to 31 characters, exclusive of spaces, tabs, and null characters, which are ignored. Allowable characters in the string are the uppercase alphabetic characters, the lowercase alphabetic characters (which are converted to uppercase), the numeric characters, the dollar sign ($), and the underscore (_).

buffer address

Address of the buffer that specifies or receives the information.

return length address

Address of a word to receive the length of information returned by $GETQUI.

$GETQUI Item Codes

QUI$_ACCOUNT_NAME

When you specify QUI$_ACCOUNT_NAME, $GETQUI returns, as a character string, the account name of the owner of the specified job. Because the account name can include up to 8 characters, the buffer length field of the item descriptor should specify 8 (bytes).

(Valid for QUI$_DISPLAY_ENTRY, QUI$_DISPLAY_JOB function codes)

QUI$_AFTER_TIME

When you specify QUI$_AFTER_TIME, $GETQUI returns, as a quadword absolute time value, the system time at or after which the specified job can execute.

(Valid for QUI$_DISPLAY_ENTRY, QUI$_DISPLAY_JOB function codes)

QUI$_ASSIGNED_QUEUE_NAME

When you specify QUI$_ASSIGNED_QUEUE_NAME, $GETQUI returns, as a character string, the name of the execution queue to which the logical queue specified in the call to $GETQUI is assigned. Because the queue name can include up to 31 characters, the buffer length field of the item descriptor should specify 31 (bytes).

(Valid for QUI$_DISPLAY_QUEUE function code)

QUI$_BASE_PRIORITY

When you specify QUI$_BASE_PRIORITY, $GETQUI returns, as a longword value in the range 0 to 15, the priority at which batch jobs are initiated from a batch execution queue or the priority of a symbiont process that controls output execution queues.

(Valid for QUI$_DISPLAY_QUEUE function code)

QUI$_CHARACTERISTIC_NAME

When you specify QUI$_CHARACTERISTIC_NAME, $GETQUI returns, as a character string, the name of the specified characteristic. Because the characteristic name can include up to 31 characters, the buffer length field of the item descriptor should specify 31 (bytes).

(Valid for QUI$_DISPLAY_CHARACTERISTIC function code)

QUI$_CHARACTERISTIC_NUMBER

When you specify QUI$_CHARACTERISTIC_NUMBER, $GETQUI returns, as a longword value in the range 0 to 127, the number of the specified characteristic.

(Valid for QUI$_DISPLAY_CHARACTERISTIC function code)

QUI$_CHARACTERISTICS

When you specify QUI$_CHARACTERISTICS, $GETQUI returns, as a 128-bit string (16-byte field), the characteristics associated with the specified queue or job. Each bit set in the bit mask represents a characteristic number in the range 0 to 127.

(Valid for QUI$_DISPLAY_ENTRY, QUI$_DISPLAY_JOB, QUI$_DISPLAY_QUEUE function codes)

QUI$_CHECKPOINT_DATA

When you specify QUI$_CHECKPOINT_DATA, $GETQUI returns, as a character string, the value of the DCL symbol BATCH$RESTART when the specified batch job is restarted. Because the value of the symbol can include up to 255 characters, the buffer length field of the item descriptor should specify 255 (bytes).

(Valid for QUI$_DISPLAY_ENTRY, QUI$_DISPLAY_JOB function codes)

QUI$_CLI

When you specify QUI$_CLI, $GETQUI returns, as an RMS file name component, the name of the command language interpreter used to execute the specified batch job. The file specification returned assumes the logical name SYS$SYSTEM and the file type EXE. Because a file name can include up to 39 characters, the buffer length field in the item descriptor should specify 39 (bytes). This item code is applicable only to batch jobs.

(Valid for QUI$_DISPLAY_ENTRY, QUI$_DISPLAY_JOB function codes)

QUI$_COMPLETED_BLOCKS

When you specify QUI$_COMPLETED_BLOCKS, $GETQUI returns, as a longword integer value, the number of blocks that the symbiont has processed for the specified print job. This item code is applicable only to print jobs.

(Valid for QUI$_DISPLAY_ENTRY, QUI$_DISPLAY_JOB function codes)

QUI$_CONDITION_VECTOR

When you specify QUI$_CONDITION_VECTOR, $GETQUI returns, as a longword condition value, the completion status of the specified job.

(Valid for QUI$_DISPLAY_ENTRY, QUI$_DISPLAY_JOB function codes)

QUI$_CPU_DEFAULT

When you specify QUI$_CPU_DEFAULT, $GETQUI returns, as a longword integer value, the default CPU time limit specified for the queue in 10-millisecond units. This item code is applicable only to batch execution queues.

For more information about default forms, see the *Guide to Maintaining a VMS System*.

(Valid for QUI$_DISPLAY_QUEUE function code)

QUI$_CPU_LIMIT

When you specify QUI$_CPU_LIMIT, $GETQUI returns, as a longword integer value, the maximum CPU time limit specified for the specified job or queue in 10-millisecond units. This item code is applicable only to batch jobs and batch execution queues.

(Valid for QUI$_DISPLAY_ENTRY, QUI$_DISPLAY_JOB, QUI$_DISPLAY_QUEUE function codes)

QUI$_DEFAULT_FORM_NAME

When you specify QUI$_DEFAULT_FORM_NAME, $GETQUI returns, as a character string, the name of the default form associated with the specified output queue. Because the form name can include up to 31 characters, the buffer length field of the item descriptor should specify 31 (bytes).

For more information about default forms, see the *Guide to Maintaining a VMS System*.

(Valid for QUI$_DISPLAY_QUEUE function code)

QUI$_DEFAULT_FORM_STOCK
When you specify QUI$_DEFAULT_FORM_STOCK, $GETQUI returns, as a character string, the name of the paper stock on which the specified default form is to be printed. Because the name of the paper stock can include up to 31 characters, the buffer length field of the item descriptor should specify 31 (bytes).

For more information on default forms, see the *Guide to Maintaining a VMS System*.

(Valid for QUI$_DISPLAY_QUEUE function code)

QUI$_DEVICE_NAME
When you specify QUI$_DEVICE_NAME, $GETQUI returns, as a character string, the name of the device on which the specified output execution queue is located. Because the device name can include up to 31 characters, the buffer length field of the item descriptor should specify 31 (bytes).

(Valid for QUI$_DISPLAY_QUEUE function code)

QUI$_ENTRY_NUMBER
When you specify QUI$_ENTRY_NUMBER, $GETQUI returns, as a longword integer value, the queue entry number of the specified job.

(Valid for QUI$_DISPLAY_ENTRY, QUI$_DISPLAY_JOB function codes)

QUI$_EXECUTING_JOB_COUNT
When you specify QUI$_EXECUTING_JOB_COUNT, $GETQUI returns, as a longword integer value, the number of jobs in the queue that are currently executing.

(Valid for QUI$_DISPLAY_QUEUE function code)

QUI$_FILE_COPIES
When you specify QUI$_FILE_COPIES, $GETQUI returns the number of times the specified file is to be processed, which is a longword integer value in the range 1 to 255. This item code is applicable only to output execution queues.

(Valid for QUI$_DISPLAY_FILE function code)

QUI$_FILE_COPIES_DONE
When you specify QUI$_FILE_COPIES_DONE, $GETQUI returns the number of times the specified file has been processed, which is a longword integer value in the range 1 to 255. This item code is applicable only to output execution queues.

(Valid for QUI$_DISPLAY_FILE function code)

QUI$_FILE_FLAGS
When you specify QUI$_FILE_FLAGS, $GETQUI returns, as a longword bit vector, the processing options that have been selected for the specified file. Each processing option is represented by a bit. When $GETQUI sets a bit, the file is processed according to the corresponding processing option. Each bit in

the vector has a symbolic name. The $QUIDEF macro defines the following symbolic names:

Symbolic Name	Description
QUI$V_FILE_BURST	Burst and flag pages are to be printed preceding the file.
QUI$V_FILE_DELETE	File is to be deleted after execution of request.
QUI$V_FILE_DOUBLE_SPACE	Symbiont formats the file with double spacing.
QUI$V_FILE_FLAG	Flag page is to be printed preceding the file.
QUI$V_FILE_TRAILER	Trailer page is to be printed following the file.
QUI$V_FILE_PAGE_HEADER	Page header is to be printed on each page of output.
QUI$V_FILE_PAGINATE	Symbiont paginates output by inserting a form feed whenever output reaches the bottom margin of the form.
QUI$V_FILE_PASSALL	Symbiont prints the file in PASSALL mode.

(Valid for QUI$_DISPLAY_FILE function code)

QUI$_FILE_IDENTIFICATION

When you specify QUI$_FILE_IDENTIFICATION, $GETQUI returns, as a 28-byte string, the internal file-identification value that uniquely identifies the selected file. This string contains (in order) the following three file-identification fields from the RMS NAM block for the selected file: the 16-byte NAM$T_DVI field, the 6-byte NAM$W_FID field, and the 6-byte NAM$W_DID field.

(Valid for QUI$_DISPLAY_FILE function code)

QUI$_FILE_SETUP_MODULES

When you specify QUI$_FILE_SETUP_MODULES, $GETQUI returns, as a comma-separated list, the names of the text modules that are to be extracted from the device control library and copied to the printer before the specified file is printed. Because a text module name can include up to 31 characters and is separated from the previous text module name with a comma, the buffer length field of the item descriptor should specify 32 (bytes) for each possible text module. This item code is meaningful only for output execution queues.

(Valid for QUI$_DISPLAY_FILE function code)

QUI$_FILE_SPECIFICATION

When you specify QUI$_FILE_SPECIFICATION, $GETQUI returns the fully qualified RMS file specification of the file about which $GETQUI is returning information. Because a file specification can include up to 255 characters, the buffer length field of the item descriptor should specify 255 (bytes).

Note: The file specification is the result of an RMS file-passing operation that occurs at the time you submit the job. If you renamed the file or created the job as a result of copying a file to a spooled device, then you cannot

use this file specification to access the file through RMS. You use QUI$_FILE_IDENTIFICATION to obtain a unique identifier for the file.

(Valid for QUI$_DISPLAY_FILE function code)

QUI$_FILE_STATUS

When you specify QUI$_FILE_STATUS, $GETQUI returns file status information as a longword bit vector. Each file status condition is represented by a bit. When $GETQUI sets the bit, the file status corresponds to the condition represented by the bit. Each bit in the vector has a symbolic name. The $QUIDEF macro defines the following symbolic names.

Symbolic Name	Description
QUI$V_FILE_CHECKPOINTED	File is checkpointed.
QUI$V_FILE_EXECUTING	File is being processed.

(Valid for QUI$_DISPLAY_FILE function code)

QUI$_FIRST_PAGE

When you specify QUI$_FIRST_PAGE, $GETQUI returns, as a longword integer value, the page number at which the printing of the specified file is to begin. This item code is applicable only to output execution queues.

(Valid for QUI$_DISPLAY_FILE function code)

QUI$_FORM_DESCRIPTION

When you specify QUI$_FORM_DESCRIPTION, $GETQUI returns, as a character string, the text string that describes the specified form. Because the text string can include up to 255 characters, the buffer length field in the item descriptor should specify 255 (bytes).

(Valid for QUI$_DISPLAY_FORM function code)

QUI$_FORM_FLAGS

When you specify QUI$_FORM_FLAGS, $GETQUI returns, as a longword bit vector, the processing options that have been selected for the specified form. Each processing option is represented by a bit. When $GETQUI sets a bit, the form is processed according to the corresponding processing option. Each bit in the vector has a symbolic name. The $QUIDEF macro defines the following symbolic names.

Symbolic Name	Description
QUI$V_FORM_SHEET_FEED	Symbiont pauses at the end of each physical page so that another sheet of paper can be inserted.
QUI$V_FORM_TRUNCATE	Printer discards any characters that exceed the specified right margin.
QUI$V_FORM_WRAP	Printer prints any characters that exceed the specified right margin on the following line.

(Valid for QUI$_DISPLAY_FORM function code)

QUI$_FORM_LENGTH

When you specify QUI$_FORM_LENGTH, $GETQUI returns, as a longword integer value, the physical length of the specified form in lines. This item code is applicable only to output execution queues.

(Valid for QUI$_DISPLAY_FORM function code)

QUI$_FORM_MARGIN_BOTTOM

When you specify QUI$_FORM_MARGIN_BOTTOM, $GETQUI returns, as a longword integer value, the bottom margin of the specified form in lines.

(Valid for QUI$_DISPLAY_FORM function code)

QUI$_FORM_MARGIN_LEFT

When you specify QUI$_FORM_MARGIN_LEFT, $GETQUI returns, as a longword integer value, the left margin of the specified form in characters.

(Valid for QUI$_DISPLAY_FORM function code)

QUI$_FORM_MARGIN_RIGHT

When you specify QUI$_FORM_MARGIN_RIGHT, $GETQUI returns, as a longword integer value, the right margin of the specified form in characters.

(Valid for QUI$_DISPLAY_FORM function code)

QUI$_FORM_MARGIN_TOP

When you specify QUI$_FORM_MARGIN_TOP, $GETQUI returns, as a longword integer value, the top margin of the specified form in lines.

(Valid for QUI$_DISPLAY_FORM function code)

QUI$_FORM_NAME

When you specify QUI$_FORM_NAME, $GETQUI returns, as a character string, the name of the specified form or the mounted form associated with the specified job or queue. Because the form name can include up to 31 characters, the buffer length field of the item descriptor should specify 31 (bytes).

For more information about mounted forms, see the *Guide to Maintaining a VMS System*.

(Valid for QUI$_DISPLAY_ENTRY, QUI$_DISPLAY_FORM, QUI$_DISPLAY_JOB, QUI$_DISPLAY_QUEUE function codes)

QUI$_FORM_NUMBER

When you specify QUI$_FORM_NUMBER, $GETQUI returns, as a longword integer value, the number of the specified form.

(Valid for QUI$_DISPLAY_FORM function code)

QUI$_FORM_SETUP_MODULES

When you specify QUI$_FORM_SETUP_MODULES, $GETQUI returns, as a comma-separated list, the names of the text modules that are to be extracted from the device control library and copied to the printer before a file is printed on the specified form. Because a text module name can include up to 31 characters and is separated from the previous text module name by a comma, the buffer length field of the item descriptor should specify 32 (bytes) for each possible text module. This item code is meaningful only for output execution queues.

(Valid for QUI$_DISPLAY_FORM function code)

QUI$_FORM_STOCK

When you specify QUI$_FORM_STOCK, $GETQUI returns, as a character string, the name of the paper stock on which the specified form is to be printed. Because the name of the paper stock can include up to 31 characters, the buffer length field of the item descriptor should specify 31 (bytes).

For more information about forms, see the *Guide to Maintaining a VMS System.*

(Valid for QUI$_DISPLAY_ENTRY, QUI$_DISPLAY_FORM, QUI$_DISPLAY_JOB, QUI$_DISPLAY_QUEUE function codes)

QUI$_FORM_WIDTH

When you specify QUI$_FORM_WIDTH, $GETQUI returns, as a longword integer value, the width of the specified form in characters.

(Valid for QUI$_DISPLAY_FORM function code)

QUI$_GENERIC_TARGET

When you specify QUI$_GENERIC_TARGET, $GETQUI returns, as a comma-separated list, the names of the execution queues that are enabled to accept work from the specified generic queue. Because a queue name can include up to 31 characters and is separated from the previous queue name with a comma, the buffer length field of the item descriptor should specify 32 (bytes) for each possible queue name. A generic queue can send work to up to 124 execution queues. This item code is meaningful only for generic queues.

(Valid for QUI$_DISPLAY_QUEUE function code)

QUI$_HOLDING_JOB_COUNT

When you specify QUI$_HOLDING_JOB_COUNT, $GETQUI returns, as a longword integer value, the number of jobs in the queue being held until explicitly released.

(Valid for QUI$_DISPLAY_QUEUE function code)

QUI$_INTERVENING_BLOCKS

When you specify QUI$_INTERVENING_BLOCKS, $GETQUI returns, as a longword integer value, the number of blocks to be processed before the specified job can begin to execute. This item code is meaningful only for output execution queues.

(Valid for QUI$_DISPLAY_ENTRY, QUI$_DISPLAY_JOB function codes)

QUI$_INTERVENING_JOBS

When you specify QUI$_INTERVENING_JOBS, $GETQUI returns, as a longword integer value, the number of jobs to be processed before the specified job can begin to execute. This item code is meaningful only for output execution queues.

(Valid for QUI$_DISPLAY_ENTRY, QUI$_DISPLAY_JOB function codes)

QUI$_JOB_COPIES

When you specify QUI$_JOB_COPIES, $GETQUI returns, as a longword integer value, the number of times the specified print job is to be repeated.

(Valid for QUI$_DISPLAY_ENTRY, QUI$_DISPLAY_JOB function codes)

QUI$_JOB_COPIES_DONE

When you specify QUI$_JOB_COPIES_DONE, $GETQUI returns, as a longword integer value, the number of times the specified print job has been repeated.

(Valid for QUI$_DISPLAY_ENTRY, QUI$_DISPLAY_JOB function codes)

QUI$_JOB_FLAGS

When you specify QUI$_JOB_FLAGS, $GETQUI returns, as a longword bit vector, the processing options that have been selected for the specified job. Each processing option is represented by a bit. When $GETQUI sets a bit, the job is processed according to the corresponding processing option. Each bit in the vector has a symbolic name. The $QUIDEF macro defines the following symbolic names.

Symbolic Name	Description
QUI$V_JOB_CPU_LIMIT	CPU time limit for the job.
QUI$V_JOB_FILE_BURST	Burst and flag pages precede each file in the job.
QUI$V_JOB_FILE_BURST_ONE	Burst and flag pages precede only the first copy of the first file in the job.
QUI$V_JOB_FILE_FLAG	Flag page precedes each file in the job.
QUI$V_JOB_FILE_FLAG_ONE	Flag page precedes only the first copy of the first file in the job.
QUI$V_JOB_FILE_PAGINATE	Symbiont paginates output by inserting a form feed whenever output reaches the bottom margin of the form.
QUI$V_JOB_FILE_TRAILER	Trailer page follows each file in the job.
QUI$V_JOB_FILE_TRAILER_ONE	Trailer page follows only the last copy of the last file in the job.
QUI$V_JOB_LOG_DELETE	Log file is deleted after it is printed.
QUI$V_JOB_LOG_NULL	No log file is created.
QUI$V_JOB_LOG_SPOOL	Job log file is queued for printing when job is complete.

Symbolic Name	Description
QUI$V_JOB_LOWERCASE	Job is to be printed on printer that can print both uppercase and lowercase letters.
QUI$V_JOB_NOTIFY	Message is broadcast to terminal when job completes or aborts.
QUI$V_JOB_RESTART	Job will restart after a system failure or can be requeued during execution.
QUI$V_JOB_WSDEFAULT	Default working set size is specified for the job.
QUI$V_JOB_WSEXTENT	Working set extent is specified for the job.
QUI$V_JOB_WSQUOTA	Working set quota is specified for the job.

(Valid for QUI$_DISPLAY_ENTRY, QUI$_DISPLAY_JOB function codes)

QUI$_JOB_LIMIT

When you specify QUI$_JOB_LIMIT, $GETQUI returns the number of jobs that can execute simultaneously on the specified queue, which is a longword integer value in the range 1 to 255. This item code is applicable only to batch execution queues.

(Valid for QUI$_DISPLAY_QUEUE function code)

QUI$_JOB_NAME

When you specify QUI$_JOB_NAME, $GETQUI returns, as a character string, the name of the specified job. Because the job name can include up to 39 characters, the buffer length field of the item descriptor should specify 39 (bytes).

(Valid for QUI$_DISPLAY_ENTRY, QUI$_DISPLAY_JOB function codes)

QUI$_JOB_PID

When you specify QUI$_JOB_PID, $GETQUI returns the process identification (PID) of the executing batch job in standard longword format.

(Valid for QUI$_DISPLAY_ENTRY, QUI$_DISPLAY_JOB function codes)

QUI$_JOB_RESET_MODULES

When you specify QUI$_JOB_RESET_MODULES, $GETQUI returns, as a comma-separated list, the names of the text modules that are to be extracted from the device control library and copied to the printer before each job in the specified queue is printed. Because a text module name can include up to 31 characters and is separated from the previous text module name by a comma, the buffer length field of the item descriptor should specify 32 (bytes) for each possible text module. This item code is meaningful only for output execution queues.

(Valid for QUI$_DISPLAY_QUEUE function code)

QUI$_JOB_SIZE

When you specify QUI$_JOB_SIZE, $GETQUI returns, as a longword integer value, the total number of disk blocks in the specified print job.

(Valid for QUI$_DISPLAY_ENTRY, QUI$_DISPLAY_JOB function codes)

QUI$_JOB_SIZE_MAXIMUM

When you specify QUI$_JOB_SIZE_MAXIMUM, $GETQUI returns, as a longword integer value, the maximum number of disk blocks that a print job initiated from the specified queue can contain. This item code is applicable only to output execution queues.

(Valid for QUI$_DISPLAY_QUEUE function code)

QUI$_JOB_SIZE_MINIMUM

When you specify QUI$_JOB_SIZE_MINIMUM, $GETQUI returns, as a longword integer value, the minimum number of disk blocks that a print job initiated from the specified queue can contain. This item code is applicable only to output execution queues.

(Valid for QUI$_DISPLAY_QUEUE function code)

QUI$_JOB_STATUS

When you specify QUI$_JOB_STATUS, $GETQUI returns the specified job's status flags, which are contained in a longword bit vector. The $QUIDEF macro defines the following symbolic names for these flags.

Symbolic Name	Description
QUI$V_JOB_ABORTING	System is attempting to abort execution of job.
QUI$V_JOB_EXECUTING	Job is executing or printing.
QUI$V_JOB_HOLDING	Job will be held until it is explicitly released.
QUI$V_JOB_INACCESSIBLE	Caller does not have READ access to the specific job and file information in the system queue file. Therefore, the QUI$_DISPLAY_JOB and QUI$_DISPLAY_FILE operations can return information for only the following output value item codes: QUI$_AFTER_TIME QUI$_COMPLETED_BLOCKS QUI$_ENTRY_NUMBER QUI$_INTERVENING_BLOCKS QUI$_INTERVENING_JOBS QUI$_JOB_SIZE QUI$_JOB_STATUS.
QUI$V_JOB_PENDING	Job is pending. See QUI$_PENDING_JOB_REASON for the reason the job is in a pending state.
QUI$V_JOB_REFUSED	Job was refused by symbiont and is waiting for symbiont to accept it for processing.
QUI$V_JOB_RETAINED	Job has completed, but it is being retained in the queue.

Symbolic Name	Description
QUI$V_JOB_STARTING	Job controller is starting to process the job and has begun communicating with an output symbiont or a job controller on another node in the cluster.
QUI$V_JOB_SUSPENDED	Job is suspended.
QUI$V_JOB_TIMED_RELEASE	Job is waiting for specified time to execute.

(Valid for QUI$_DISPLAY_ENTRY, QUI$_DISPLAY_JOB function codes)

QUI$_LAST_PAGE

When you specify QUI$_LAST_PAGE, $GETQUI returns, as a longword integer value, the page number at which the printing of the specified file should end. This item code is applicable only to output execution queues.

(Valid for QUI$_DISPLAY_FILE function code)

QUI$_LIBRARY_SPECIFICATION

When you specify QUI$_LIBRARY_SPECIFICATION, $GETQUI returns, as an RMS file name component, the name of the device control library for the specified queue. The library specification assumes the device and directory name SYS$LIBRARY and a file type of TLB. Because a file name can include up to 39 characters, the buffer length field of the item descriptor should specify 39 (bytes). This item code is meaningful only for output execution queues.

(Valid for QUI$_DISPLAY_QUEUE function code)

QUI$_LOG_QUEUE

When you specify QUI$_LOG_QUEUE, $GETQUI returns, as a character string, the name of the queue into which the log file produced for the specified batch job is to be entered for printing. This item code is applicable only to batch jobs. Because a queue name can contain up to 31 characters, the buffer length field of the item descriptor should specify 31 (bytes).

(Valid for QUI$_DISPLAY_ENTRY, QUI$_DISPLAY_JOB function codes)

QUI$_LOG_SPECIFICATION

When you specify QUI$_LOG_SPECIFICATION, $GETQUI returns, as an RMS file specification, the name of the log file to be produced for the specified job. Because a file specification can include up to 255 characters, the buffer length field of the item descriptor should specify 255 (bytes). This item code is meaningful only for batch jobs.

The string returned is the log file specification that was provided to the $SNDJBC service to create the job. Therefore, to determine whether a log file is to be produced, testing this item code for a zero-length string is insufficient; instead, you need to examine the QUI$V_JOB_LOG_NULL bit of the QUI$_JOB_FLAGS item code.

(Valid for QUI$_DISPLAY_ENTRY, QUI$_DISPLAY_JOB function codes)

QUI$_NOTE

When you specify QUI$_NOTE, $GETQUI returns, as a character string, the note that is to be printed on the job flag and file flag pages of the specified job. Because the note can include up to 255 characters, the buffer length field of the item descriptor should specify 255 (bytes). This item code is meaningful only for output execution queues.

(Valid for QUI$_DISPLAY_ENTRY, QUI$_DISPLAY_JOB function codes)

QUI$_OPERATOR_REQUEST

When you specify QUI$_OPERATOR_REQUEST, $GETQUI returns, as a character string, the message that is to be sent to the queue operator before the specified job begins to execute. Because the message can include up to 255 characters, the buffer length field of the item descriptor should specify 255 (bytes). This item code is meaningful only for output execution queues.

(Valid for QUI$_DISPLAY_ENTRY, QUI$_DISPLAY_JOB function codes)

QUI$_OWNER_UIC

When you specify QUI$_OWNER_UIC, $GETQUI returns the owner UIC as a longword value in standard UIC format. For information on UIC format, see the Identifier Format section in the "Security Services" chapter of the *Introduction to VMS System Services.*

(Valid for QUI$_DISPLAY_QUEUE function code)

QUI$_PAGE_SETUP_MODULES

When you specify QUI$_PAGE_SETUP_MODULES, $GETQUI returns, as a comma-separated list, the names of the text modules to be extracted from the device control library and copied to the printer before each page of the specified form is printed. Because a text module name can include up to 31 characters and is separated from the previous text module name by a comma, the buffer length field of the item descriptor should specify 32 (bytes) for each possible text module. This item code is meaningful only for output execution queues.

(Valid for QUI$_DISPLAY_FORM function code)

QUI$_PARAMETER_1 through QUI$_PARAMETER_8

When you specify QUI$_PARAMETER_1 through QUI$_PARAMETER_8, $GETQUI returns, as a character string, the value of the user-defined parameters, that in batch jobs become the value of the DCL symbols P1 through P8, respectively. Because these parameters may include up to 255 characters, the buffer length field of the item descriptor should specify 255 (bytes).

(Valid for QUI$_DISPLAY_ENTRY, QUI$_DISPLAY_JOB function codes)

QUI$_PENDING_JOB_BLOCK_COUNT

When you specify QUI$_PENDING_JOB_BLOCK_COUNT, $GETQUI returns, as a longword integer value, the total number of blocks for all pending jobs in the queue (valid only for output execution queues).

(Valid for QUI$_DISPLAY_QUEUE function code)

QUI$_PENDING_JOB_COUNT

When you specify QUI$_PENDING_JOB_COUNT, $GETQUI returns, as a longword integer value, the number of jobs in the queue in a pending state.

(Valid for QUI$_DISPLAY_QUEUE function code)

QUI$_PENDING_JOB_REASON

When you specify QUI$_PENDING_JOB_REASON, $GETQUI returns, as a longword bit vector, the reason that the job is in a pending state. The $QUIDEF macro defines the following symbolic names for the flags.

Symbolic Name	Description
QUI$V_PEND_CHAR_MISMATCH	Job requires characteristics that are not available on the execution queue.
QUI$V_PEND_JOB_SIZE_MAX	Block size of job exceeds the upper block limit of the execution queue.
QUI$V_PEND_JOB_SIZE_MIN	Block size of job is less than the lower limit of the execution queue.
QUI$V_PEND_LOWERCASE_MISMATCH	Job requires lowercase printer.
QUI$V_PEND_NO_ACCESS	Owner of job does not have access to the execution queue.
QUI$V_PEND_QUEUE_BUSY	Job is pending because the number of jobs currently executing on the queue equals the job limit for the queue.
QUI$V_PEND_QUEUE_STATE	Job is pending because the execution queue is not in a running, open state as indicated by QUI$_QUEUE_STATUS.
QUI$V_PEND_STOCK_MISMATCH	Stock type required by the job's form does not match the stock type of the form mounted on the execution queue.

(Valid for QUI$_DISPLAY_ENTRY, QUI$_DISPLAY_JOB function codes)

QUI$_PRIORITY

When you specify QUI$_PRIORITY, $GETQUI returns the scheduling priority of the specified job, which is a longword integer value in the range 0 through 255.

Scheduling priority affects the order in which jobs assigned to a queue are initiated; it has no effect on the base execution priority of a job. The lowest scheduling priority value is 0, the highest is 255; that is, if a queue contains a job with a scheduling priority of 10 and a job with a scheduling priority of 2, the queue manager initiates the job with the scheduling priority of 10 first.

(Valid for QUI$_DISPLAY_ENTRY, QUI$_DISPLAY_JOB function codes)

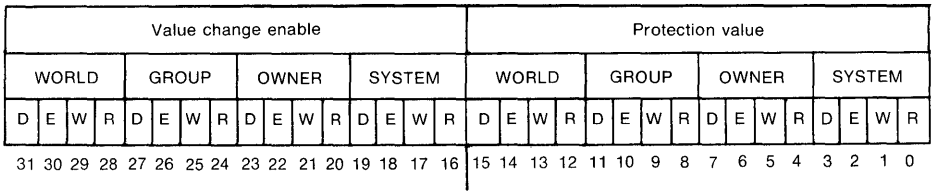
QUI$_PROCESSOR

When you specify QUI$_PROCESSOR, $GETQUI returns, as an RMS file name component, the name of the symbiont image that executes print jobs initiated from the specified queue. The file name assumes the device and directory name SYS$SYSTEM and the file type EXE. Because an RMS file name can include up to 39 characters, the buffer length field of the item descriptor should specify 39 (bytes).

(Valid for QUI$_DISPLAY_QUEUE function code)

QUI$_PROTECTION

When you specify QUI$_PROTECTION, $GETQUI returns, as a longword, the specified queue's protection mask. The following diagram depicts the protection mask.



ZK-1724-84

Bits 0 through 15 specify the protection value: the four types of access (read, write, execute, delete) to be granted to the four classes of user (system, owner, group, world). Set bits deny access and clear bits allow access.

Bits 16 through 31 enable or disable bits 0 through 15. When you set a bit in the second word, the corresponding bit in the first word affects the queue protection. When you clear a bit in the second word, the corresponding bit in the first word is ignored.

By default, the queue protection is (S:E,O:D,G:R,W:W).

(Valid for QUI$_DISPLAY_QUEUE function code)

QUI$_QUEUE_DESCRIPTION

When you specify QUI$_QUEUE_DESCRIPTION, $GETQUI returns, as a character string, the text that describes the specified queue. Because the text can include up to 255 characters, the buffer length field of the item descriptor should specify 255 (bytes).

(Valid for QUI$_DISPLAY_QUEUE function code)

QUI$_QUEUE_FLAGS

When you specify QUI$_QUEUE_FLAGS, $GETQUI returns, as a longword bit vector, the processing options that have been selected for the specified queue. Each processing option is represented by a bit. When $GETQUI sets a bit, the jobs initiated from the queue are processed according to the corresponding processing option. Each bit in the vector has a symbolic name. The $QUIDEF macro defines the following symbolic names.

Symbolic Name	Description
QUI$V_QUEUE_ACL_SPECIFIED	An access control list has been specified for the queue. You cannot retrieve a queue's ACL through the $GETQUI service. Instead, you must use the $CHANGE_ACL service.
QUI$V_QUEUE_BATCH	Queue is a batch queue or a generic batch queue.
QUI$V_QUEUE_CPU_DEFAULT	A default CPU time limit has been specified for all jobs in the queue.
QUI$V_QUEUE_CPU_LIMIT	A maximum CPU time limit has been specified for all jobs in the queue.
QUI$V_QUEUE_FILE_BURST	Burst and flag pages precede each file in each job initiated from the queue.
QUI$V_QUEUE_FILE_BURST_ONE	Burst and flag pages precede only the first copy of the first file in each job initiated from the queue.
QUI$V_QUEUE_FILE_FLAG	Flag page precedes each file in each job initiated from the queue.
QUI$V_QUEUE_FILE_FLAG_ONE	Flag page precedes only the first copy of the first file in each job initiated from the queue.
QUI$V_QUEUE_FILE_PAGINATE	Output symbiont paginates output for each job initiated from this queue. The output symbiont paginates output by inserting a form feed whenever output reaches the bottom margin of the form.
QUI$V_QUEUE_FILE_TRAILER	Trailer page follows each file in each job initiated from the queue.
QUI$V_QUEUE_FILE_TRAILER_ONE	Trailer page follows only the last copy of the last file in each job initiated from the queue.
QUI$V_QUEUE_GENERIC	The queue is a generic queue.
QUI$V_QUEUE_GENERIC_SELECTION	The queue is an execution queue that can accept work from a generic queue.
QUI$V_QUEUE_JOB_BURST	Burst and flag pages precede each job initiated from the queue.
QUI$V_QUEUE_JOB_FLAG	A flag page precedes each job initiated from the queue.
QUI$V_QUEUE_JOB_SIZE_SCHED	Jobs initiated from the queue are scheduled according to size, with the smallest job of a given priority processed first (meaningful only for output queues).
QUI$V_QUEUE_JOB_TRAILER	A trailer page follows each job initiated from the queue.
QUI$V_QUEUE_PRINTER	The queue is a printer queue.

Symbolic Name	Description
QUI$V_QUEUE_RECORD_BLOCKING	The symbiont is permitted to concatenate, or block together, the output records it sends to the output device.
QUI$V_QUEUE_RETAIN_ALL	All jobs initiated from the queue remain in the queue after they finish executing. Completed jobs are marked with a completion status.
QUI$V_QUEUE_RETAIN_ERROR	Only jobs that do not complete successfully are retained in the queue.
QUI$V_QUEUE_SWAP	Jobs initiated from the queue can be swapped.
QUI$V_QUEUE_TERMINAL	The queue is a terminal queue.
QUI$V_QUEUE_WSDEFAULT	Default working set size is specified for each job initiated from the queue.
QUI$V_QUEUE_WSEXTENT	Working set extent is specified for each job initiated from the queue.
QUI$V_QUEUE_WSQUOTA	Working set quota is specified for each job initiated from the queue.

(Valid for QUI$_DISPLAY_QUEUE function code)

QUI$_QUEUE_NAME

When you specify QUI$_QUEUE_NAME, $GETQUI returns, as a character string, the name of the specified queue or the name of the queue that contains the specified job. Because a queue name can include up to 31 characters, the buffer length field of the item descriptor should specify 31 (bytes).

(Valid for QUI$_DISPLAY_ENTRY, QUI$_DISPLAY_JOB, QUI$_DISPLAY_QUEUE function codes)

QUI$_QUEUE_STATUS

When you specify QUI$_QUEUE_STATUS, $GETQUI returns the specified queue's status flags, which are contained in a longword bit vector. The $QUIDEF macro defines the following symbolic names for these flags.

Symbolic Name	Description
QUI$V_QUEUE_ALIGNING	Queue is printing alignment pages.
QUI$V_QUEUE_CLOSED	Queue is closed and will not accept new jobs until the queue is put in an open state.
QUI$V_QUEUE_IDLE	Queue contains no job requests.
QUI$V_QUEUE_LOWERCASE	Queue is associated with a printer that can print both uppercase and lowercase characters.
QUI$V_QUEUE_PAUSED	Execution of all current jobs in the queue is temporarily halted.

Symbolic Name	Description
QUI$V_QUEUE_PAUSING	Queue is temporarily halting execution. Currently executing jobs are completing; no new jobs can begin executing.
QUI$V_QUEUE_REMOTE	Queue is assigned to a physical device that is not connected to the local node.
QUI$V_QUEUE_RESETTING	Queue is resetting and stopping.
QUI$V_QUEUE_RESUMING	Queue is restarting after pausing.
QUI$V_QUEUE_SERVER	Queue processing is directed to a server symbiont.
QUI$V_QUEUE_STALLED	Physical device to which queue is assigned is stalled; that is, the device has not completed the last I/O request submitted to it.
QUI$V_QUEUE_STARTING	Queue is starting.
QUI$V_QUEUE_STOPPED	Queue is stopped.
QUI$V_QUEUE_STOPPING	Queue is stopping.
QUI$V_QUEUE_UNAVAILABLE	Physical device to which queue is assigned is not available.

(Valid for QUI$_DISPLAY_QUEUE function code)

QUI$_REQUEUE_QUEUE_NAME

When you specify QUI$_REQUEUE_QUEUE_NAME, $GETQUI returns, as a character string, the name of the queue to which the specified job is reassigned. Because a queue name can contain up to 31 characters, the buffer length field of the item descriptor should specify 31 (bytes).

(Valid for QUI$_DISPLAY_ENTRY, QUI$_DISPLAY_JOB function codes)

QUI$_RESTART_QUEUE_NAME

When you specify QUI$_RESTART_QUEUE_NAME, $GETQUI returns, as a character string, the name of the queue in which the job will be placed if the job is restarted.

(Valid for QUI$_DISPLAY_JOB function code)

QUI$_RETAINED_JOB_COUNT

When you specify QUI$_RETAINED_JOB_COUNT, $GETQUI returns, as a longword integer value, the number of jobs in the queue retained after successful completion plus those retained on error.

(Valid for QUI$_DISPLAY_QUEUE function code)

QUI$_SCSNODE_NAME

When you specify QUI$_SCSNODE_NAME, $GETQUI returns, as a character string, the name of the VAX node on which the specified execution queue is located. Because the node name can include up to 6 characters, the buffer length field of the item descriptor should specify 6 (bytes).

(Valid for QUI$_DISPLAY_QUEUE function code)

QUI$_SEARCH_FLAGS

When you specify QUI$_SEARCH_FLAGS, an input value item code, it specifies a longword bit vector wherein each bit specifies the scope of $GETQUI's search for objects specified in the call to $GETQUI. The $QUIDEF macro defines symbols for each option (bit) in the bit vector. The following table contains the symbolic names for each option and the function code for which each flag is meaningful.

Symbolic Name	Function Code	Description
QUI$V_FREEZE_CONTEXT	QUI$_DISPLAY_CHARACTERISTIC QUI$_DISPLAY_ENTRY QUI$_DISPLAY_FILE QUI$_DISPLAY_FORM QUI$_DISPLAY_JOB QUI$_DISPLAY_QUEUE	Does not advance wildcard context on completion of this service call.
QUI$V_SEARCH_ALL_JOBS	QUI$_DISPLAY_JOB	$GETQUI searches all jobs included in the established queue context. If you do not specify this flag, $GETQUI only returns information about jobs that have the same user name as the caller.
QUI$V_SEARCH_BATCH	QUI$_DISPLAY_ENTRY QUI$_DISPLAY_QUEUE	Selects batch queues.
QUI$V_SEARCH_EXECUTING_JOBS	QUI$_DISPLAY_ENTRY QUI$_DISPLAY_JOB	Selects executing jobs.
QUI$V_SEARCH_GENERIC	QUI$_DISPLAY_QUEUE	Selects generic queues.
QUI$V_SEARCH_HOLDING_JOBS	QUI$_DISPLAY_ENTRY QUI$_DISPLAY_JOB	Selects jobs on unconditional hold.
QUI$V_SEARCH_PENDING_JOBS	QUI$_DISPLAY_ENTRY QUI$_DISPLAY_JOB	Selects pending jobs.
QUI$V_SEARCH_PRINTER	QUI$_DISPLAY_ENTRY QUI$_DISPLAY_QUEUE	Selects printer queues.
QUI$V_SEARCH_RETAINED_JOBS	QUI$_DISPLAY_ENTRY QUI$_DISPLAY_JOB	Selects jobs being retained.
QUI$V_SEARCH_SERVER	QUI$_DISPLAY_ENTRY QUI$_DISPLAY_QUEUE	Selects server queues.
QUI$V_SEARCH_SYMBIONT	QUI$_DISPLAY_ENTRY QUI$_DISPLAY_QUEUE	Selects output queues.
QUI$V_SEARCH_TERMINAL	QUI$_DISPLAY_ENTRY QUI$_DISPLAY_QUEUE	Selects terminal queues.

Symbolic Name	Function Code	Description
QUI$V_SEARCH_THIS_JOB	QUI$_DISPLAY_FILE QUI$_DISPLAY_JOB QUI$_DISPLAY_QUEUE	$GETQUI returns information about the calling batch job, the command file being executed, or the queue associated with the calling batch job. $GETQUI establishes a new queue and job context based on the job entry of the caller; this queue and job context is dissolved when $GETQUI finishes executing. If you specify QUI$V_SEARCH_THIS_JOB, $GETQUI ignores all other QUI$_SEARCH_FLAGS options.
QUI$V_SEARCH_TIMED_RELEASE_JOBS	QUI$_DISPLAY_ENTRY QUI$_DISPLAY_JOB	Selects jobs on hold until a specified time.
QUI$V_SEARCH_WILDCARD	QUI$_DISPLAY_CHARACTERISTIC QUI$_DISPLAY_ENTRY QUI$_DISPLAY_FORM QUI$_DISPLAY_QUEUE	$GETQUI performs a search in wildcard mode even if QUI$_SEARCH_NAME contains no wildcard characters.

QUI$_SEARCH_NAME

QUI$_SEARCH_NAME is an input value item code, which specifies, as a 1- to 31-character string, the name of the object about which $GETQUI is to return information. The buffer must specify the name of a characteristic, form, or queue.

To direct $GETQUI to perform a wildcard search, you specify QUI$_SEARCH_NAME as a string containing one or more of the wildcard characters (% or *).

(Valid for QUI$_DISPLAY_CHARACTERISTIC, QUI$_DISPLAY_FORM, QUI$_DISPLAY_QUEUE function codes)

QUI$_SEARCH_NUMBER

QUI$_SEARCH_NUMBER is an input value item code, which specifies, as a longword integer value, the number of the characteristic, form, or job entry about which $GETQUI is to return information. The buffer must specify a longword integer value.

(Valid for QUI$_DISPLAY_CHARACTERISTIC, QUI$_DISPLAY_ENTRY, QUI$_DISPLAY_FORM function codes)

QUI$_SEARCH_USERNAME

QUI$_SEARCH_USERNAME is an input value item code, which specifies, as a 1- to 12-character string, the user name for $GETQUI to use to restrict its search for jobs. By default, $GETQUI searches for jobs whose owner has the same user name as the calling process.

(Valid for QUI$_DISPLAY_ENTRY function code)

QUI$_SUBMISSION_TIME

When you specify QUI$_SUBMISSION_TIME, $GETQUI returns, as a quadword absolute time value, the time at which the specified job was submitted to the queue.

(Valid for QUI$_DISPLAY_ENTRY, QUI$_DISPLAY_JOB function codes)

QUI$_TIMED_RELEASE_JOB_COUNT

When you specify QUI$_TIMED_RELEASE_JOB_COUNT, $GETQUI returns, as a longword value, the number of jobs in the queue on hold until a specified time.

(Valid for QUI$_DISPLAY_QUEUE function code)

QUI$_UIC

When you specify QUI$_UIC, $GETQUI returns, in standard longword format, the UIC of the owner of the specified job.

(Valid for QUI$_DISPLAY_ENTRY, QUI$_DISPLAY_JOB function codes)

QUI$_USERNAME

When you specify QUI$_USERNAME, $GETQUI returns, as a character string, the user name of the owner of the specified job. Because the user name can include up to 12 characters, the buffer length field of the item descriptor should specify 12 (bytes).

(Valid for QUI$_DISPLAY_ENTRY, QUI$_DISPLAY_JOB function codes)

QUI$_WSDEFAULT

When you specify QUI$_WSDEFAULT, $GETQUI returns the default working set size specified for the specified job or queue, which is a longword integer in the range 1 through 65,535. This value is meaningful only for batch jobs and execution and output queues.

(Valid for QUI$_DISPLAY_ENTRY, QUI$_DISPLAY_JOB, QUI$_DISPLAY_QUEUE function codes)

QUI$_WSEXTENT

When you specify QUI$_WSEXTENT, $GETQUI returns the working set extent specified for the specified job or queue, which is a longword integer in the range 1 through 65,535. This value is meaningful only for batch jobs and execution and output queues.

(Valid for QUI$_DISPLAY_ENTRY, QUI$_DISPLAY_JOB, QUI$_DISPLAY_QUEUE function codes)

QUI$_WSQUOTA

When you specify QUI$_WSQUOTA, $GETQUI returns the working set quota for the specified job or queue, which is a longword integer in the range 1 through 65,535. This value is meaningful only for batch jobs and execution and output queues.

(Valid for QUI$_DISPLAY_ENTRY, QUI$_DISPLAY_JOB, QUI$_DISPLAY_QUEUE function codes)

iosb

VMS usage: **io_status_block**
type: **quadword (unsigned)**
access: **write only**
mechanism: **by reference**

I/O status block into which $GETQUI writes the completion status after the requested operation has completed. The **iosb** argument is the address of the I/O status block.

At request initiation $GETQUI sets the value of the quadword I/O status block to 0. When the requested operation has completed, $GETQUI writes a condition value in the first longword of the I/O status block. It writes the value 0 into the second longword; this longword is unused and reserved for future use.

The condition values returned by $GETQUI in the I/O status block are condition values from the JBC facility, which are defined by the $JBCMSGDEF macro. The condition values returned from the JBC facility are listed under CONDITION VALUES RETURNED IN THE I/O STATUS BLOCK.

Though this argument is optional, DIGITAL strongly recommends that you specify it, for the following reasons:

- If you are using an event flag to signal the completion of the service, you can test the I/O status block for a condition value to be sure that the event flag was not set by an event other than service completion.
- If you are using the $SYNCH service to synchronize completion of the service, the I/O status block is a required argument for $SYNCH.
- The condition value returned in R0 and the condition value returned in the I/O status block provide information about different aspects of the call to the $GETQUI service. The condition value returned in R0 gives you information about the success or failure of the service call itself; the condition value returned in the I/O status block gives you information about the success or failure of the service operation. Therefore, to accurately assess the success or failure of the call to $GETQUI, you must check the condition values returned in both R0 and the I/O status block.

astadr

VMS usage: **ast_procedure**
type: **procedure entry mask**
access: **call without stack unwinding**
mechanism: **by reference**

AST service routine to be executed when $GETQUI completes. The **astadr** argument is the address of the entry mask of this routine.

If specified, the AST routine executes at the same access mode as the caller of $GETQUI.

astprm

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

AST parameter to be passed to the AST service routine specified by the **astadr** argument. The **astprm** argument is this longword parameter.

DESCRIPTION

$GETQUI's Information Search

The $GETQUI service returns information about objects defined in the system job queue file. You may specify the following function codes to return information for the object types listed:

Function Code	Object Type
QUI$_DISPLAY_CHARACTERISTIC	Characteristic
QUI$_DISPLAY_FORM	Form
QUI$_DISPLAY_QUEUE	Queue
QUI$_DISPLAY_JOB	Job within a queue context
QUI$_DISPLAY_FILE	File within a job context
QUI$_DISPLAY_ENTRY	Job independent of queue

When you call the $GETQUI service, the job controller establishes an internal GETQUI context block (GQC). The system uses the GQC to store information temporarily and to keep track of its place in a wildcard sequence of operations. The system provides only one GQC per process; therefore, only one $GETQUI operation can be in progress at any one time.

To allow you to obtain information either about a particular object in a single call or about several objects in a sequence of calls, $GETQUI supports three different search modes. The following search modes affect the disposition of the GQC in different ways:

- Nonwildcard Mode—$GETQUI returns information about a particular object in a single call. After the call completes, the system dissolves the GQC.
- Wildcard Mode—$GETQUI returns information about several objects of the same type in a sequence of calls. The system saves the GQC between calls until the wildcard sequence completes.
- Nested Wildcard Mode—$GETQUI returns information about objects defined within another object. Specifically, this mode allows you to query jobs contained in a selected queue or files contained in a selected job in a sequence of calls. After each call, the system saves the GQC so that the GQC can provide the queue or job context necessary for subsequent calls.

The sections that follow describe how each of the three search methods affects $GETQUI's search for information; how you direct $GETQUI to undertake each method; and how each method affects the contents of the GQC.

Nonwildcard Mode

In nonwildcard mode, $GETQUI can return information about the following objects:

- A specific characteristic or form definition that you identify by name or number.
- A specific queue definition that you identify by name.
- A specific batch or print job that you identify by job entry number.
- The queue, job, or executing command procedure file associated with the calling batch job. You invoke this special case of nonwildcard mode by specifying the QUI$_SEARCH_THIS_JOB option of the QUI$_SEARCH_FLAGS item code for a display queue, job, or file operation.

To obtain information about a specific characteristic or form definition, you call $GETQUI using the QUI$_DISPLAY_CHARACTERISTIC or QUI$_DISPLAY_FORM function code. You also need to specify either the name of the characteristic or form in the QUI$_SEARCH_NAME item code or the number of the characteristic or form in the QUI$_SEARCH_NUMBER item code. The name string you specify may not include either of the wildcard characters (* or %). You may specify both the QUI$_SEARCH_NAME and QUI$_SEARCH_NUMBER item codes, but the name and number you specify must be associated with the same characteristic or form definition.

To obtain information about a specific queue definition, you specify the QUI$_DISPLAY_QUEUE function code and provide the name of the queue in the QUI$_SEARCH_NAME item code. The name string you specify may not include the wildcard characters (* or %).

To obtain information about a specific batch or print job, you specify the QUI$_DISPLAY_ENTRY function code and provide the entry number of the job in the QUI$_SEARCH_NUMBER item code.

Finally, the $GETQUI service provides an option that allows a batch job to obtain information about the queue, job, or command file that the associated batch job is executing without first entering wildcard mode to establish a queue or job context. You can make a call from the batch job that specifies the QUI$_DISPLAY_QUEUE function code to obtain information about the queue from which the batch job was initiated; the QUI$_DISPLAY_JOB function code to obtain information about the batch job itself; or the QUI$_DISPLAY_FILE function code to obtain information about the command file for the batch job. For each of these calls, you must select the QUI$V_SEARCH_THIS_JOB option of the QUI$_SEARCH_FLAGS item code. When you select this option, $GETQUI ignores all other options in the QUI$_SEARCH_FLAGS item code.

Wildcard Mode

In wildcard mode, the system saves the GQC between calls to $GETQUI so that you can make a sequence of calls to $GETQUI to get information about all characteristics, form definitions, queues, or jobs contained in the system job queue file.

To set up a wildcard search for characteristic or form definitions, specify the QUI$_DISPLAY_CHARACTERISTIC or QUI$_DISPLAY_FORM function code and specify a name in the QUI$_SEARCH_NAME item code that includes one or more wildcard characters (* or %).

To set up a wildcard search for queue definitions, you specify the QUI$_DISPLAY_QUEUE function code and specify a name in the QUI$_SEARCH_NAME item code that includes one or more wildcard characters (* or %). You can indicate the type of the queue you want to search for by specifying any combination of the following options for the QUI$_SEARCH_FLAGS item code:

QUI$V_SEARCH_BATCH
QUI$V_SEARCH_PRINTER
QUI$V_SEARCH_SERVER
QUI$V_SEARCH_TERMINAL
QUI$V_SEARCH_SYMBIONT
QUI$V_SEARCH_GENERIC

For example, if you select the QUI$V_SEARCH_BATCH option, $GETQUI returns information only about batch queues; if you select the QUI$V_SEARCH_SYMBIONT option, $GETQUI returns information only about output queues (printer, terminal, and server queues). If you specify none of the queue type options, $GETQUI searches all queues.

To set up a wildcard search for jobs, you specify the QUI$_DISPLAY_ENTRY function code and the QUI$_SEARCH_WILDCARD option of the QUI$_SEARCH_FLAGS item code. When you specify this option, omit the QUI$_SEARCH_NUMBER item code. You can restrict the search to jobs having particular status, or to jobs residing in specific types of queues, or both, by including any combination of the following options for the QUI$_SEARCH_FLAGS item code:

QUI$V_SEARCH_BATCH
QUI$V_SEARCH_EXECUTING_JOBS
QUI$V_SEARCH_HOLDING_JOBS
QUI$V_SEARCH_PENDING_JOBS
QUI$V_SEARCH_PRINTER
QUI$V_SEARCH_RETAINED_JOBS
QUI$V_SEARCH_SERVER
QUI$V_SEARCH_SYMBIONT
QUI$V_SEARCH_TERMINAL
QUI$V_SEARCH_TIMED_RELEASE_JOBS

You can also force wildcard mode for characteristic, form, or queue display operations by specifying the QUI$V_SEARCH_WILDCARD option of the QUI$_SEARCH_FLAGS item code. If you specify this option, the system saves the GQC between calls, even if you specify a nonwildcard name in the QUI$_SEARCH_NAME item code. Whether you specify a wildcard name in the QUI$_SEARCH_NAME item code, selecting the QUI$V_SEARCH_WILDCARD option ensures that wildcard mode is enabled.

Once established, wildcard mode remains in effect until one of the following actions occurs, which causes the GQC to be released.

- $GETQUI returns a JBC$_NOMORExxx or JBC$_NOSUCHxxx condition value on a call to display characteristic, form, queue, or entry information, where *xxx* refers to CHAR, FORM, QUE, or ENT.

- You explicitly cancel the wildcard operation by specifying the QUI$_CANCEL_OPERATION function code in a call to the $GETQUI service.
- Your process terminates.

Note that wildcard mode is a prerequisite for entering nested wildcard mode.

Nested Wildcard Mode

In nested wildcard mode, the system saves the GQC between calls to $GETQUI so that you can make a sequence of calls to $GETQUI to get information about jobs that are contained in a selected queue or files of the selected job. Nested wildcard mode reflects the parent-child relationship between queues and jobs and between jobs and files. The $GETQUI service can locate and return information about only one object in a single call. However, queues are objects that contain jobs and jobs are objects that contain files. Therefore, to get information about an object contained within another object, you must first make a call to $GETQUI that specifies and locates the containing object and then make a call to request information about the contained object. The system saves the location of the containing object in the GQC along with the location of the contained object.

Two of $GETQUI's operations, QUI$_DISPLAY_JOB and QUI$_DISPLAY_FILE, can be used only in a nested wildcard mode, with one exception. The exceptional use of these two operations involves calls made to $GETQUI from a batch job to find out more information about itself. This exceptional use is described at the end of the Nonwildcard Mode section.

You can enter nested wildcard mode from either wildcard display queue mode or from wildcard display entry mode. To obtain job and file information in nested wildcard mode, you can use a combination of QUI$_DISPLAY_QUEUE, QUI$_DISPLAY_JOB, and QUI$_DISPLAY_FILE operations. To obtain file information, you can use a combination of QUI$_DISPLAY_ENTRY and QUI$_DISPLAY_FILE operations as an alternative.

To set up a nested wildcard search for job and file information, you first perform one or more QUI$_DISPLAY_QUEUE operations in wildcard mode to establish the queue context necessary for the nested display job and file operations. Next you specify the QUI$_DISPLAY_JOB operation repetitively; these calls search the current queue until a call locates the job that contains the file or files you want. This call establishes the job context. Having located the queue and the job that contain the file or files, you can now use the QUI$_DISPLAY_FILE operation repetitively to request file information.

You can enter the nested wildcard mode for the display queue operation in two different ways: by specifying a wildcard name in the QUI$_SEARCH_NAME item code, or by specifying a nonwildcard queue name and selecting the QUI$V_SEARCH_WILDCARD option of the QUI$_SEARCH_FLAG item code. The second method of entering wildcard mode is useful if you want to obtain information about one or more jobs or files within jobs for a specific queue and, therefore, want to specify a nonwildcard queue name, but still want to save the GQC after the queue context is established.

When you make calls to $GETQUI that specify the QUI$_DISPLAY_JOB function code, by default $GETQUI locates all the jobs in the selected queue that have the same user name as the calling process. If you want to obtain information about all the jobs in the selected queue, you select the QUI$V_SEARCH_ALL_JOBS option of the QUI$_SEARCH_FLAGS item code.

After you establish a queue context, it remains in effect until either you change the context by making another call to $GETQUI that specifies the QUI$_DISPLAY_QUEUE function code, or one of the actions listed at the end of the Wildcard Mode section causes the GQC to be released. An established job context remains in effect until you change the context by making another call to $GETQUI that specifies the QUI$_DISPLAY_JOB function code or $GETQUI returns a JBC$_NOMOREJOB or JBC$_NOSUCHJOB condition value. While the return of either of these two condition values releases the job context, the wildcard search remains in effect, because the GQC continues to maintain the queue context. Similarly, return of the JBC$_NOMOREFILE or JBC$_NOSUCHFILE condition value signals that no more files remain in the current job context. However, these condition values do not cause the job context to be dissolved.

To set up a nested wildcard search for file information for a particular entry, you first perform one or more QUI$_DISPLAY_ENTRY operations in wildcard mode to establish the desired job context. Next you call $GETQUI iteratively with the QUI$_DISPLAY_FILE function code to obtain file information for the selected job.

When you make calls to $GETQUI that specify the QUI$_DISPLAY_ENTRY function code, by default $GETQUI locates all jobs that have the same user name as the calling process. If you want to obtain information about jobs owned by another user, you specify the user name in the QUI$_SEARCH_USERNAME item code.

You can use the QUI$_FREEZE_CONTEXT option of the QUI$_SEARCH_FLAGS item code in any wildcard or nested wildcard call to prevent advancement of context to the next object on the list. This allows you to make successive calls for information about the same queue, job, file, characteristic, or form.

Privileges Required for Obtaining Job Information

The caller must have READ access to the job or SYSPRV or OPER privilege to obtain job and file information. If the caller does not have privilege to access a job specified in a QUI$_DISPLAY_JOB or QUI$_DISPLAY_FILE operation, $GETQUI returns a successful condition value. However, it sets the QUI$V_JOB_INACCESSIBLE bit of the QUI$_JOB_STATUS item code and returns information only for the following item codes:

QUI$_AFTER_TIME
QUI$_COMPLETED_BLOCKS
QUI$_ENTRY_NUMBER
QUI$_INTERVENING_BLOCKS
QUI$_INTERVENING_JOBS
QUI$_JOB_SIZE
QUI$_JOB_STATUS

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The item list or input buffer cannot be read by the caller; or the return length buffer, output buffer, or status block cannot be written by the caller.
SS$_BADPARAM	The function code is invalid; the item list contains an invalid item code; a buffer descriptor has an invalid length; or the reserved parameter has a nonzero value.
SS$_DEVOFFLINE	The job controller process is not running.
SS$_EXASTLM	The **astadr** argument was specified, and the process has exceeded its ASTLM quota.
SS$_ILLEFC	The **efn** argument specifies an illegal event flag number.
SS$_INSFMEM	The space for completing the request is insufficient.
SS$_MBFULL	The job controller mailbox is full.
SS$_MBTOOSML	The mailbox message is too large for the job controller mailbox.
SS$_UNASEFC	The **efn** argument specifies an unassociated event flag cluster.

CONDITION VALUES RETURNED IN THE I/O STATUS BLOCK

JBC$_NORMAL	The service completed successfully.
JBC$_INVFUNCOD	The specified function code is invalid.
JBC$_INVITMCOD	The item list contains an invalid item code.
JBC$_INVPARLEN	The length of a specified string is outside the valid range for that item code.
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$_MISREQPAR	An item code that is required for the specified function code has not been specified.
JBC$_NOJOBCTX	No job context has been established for a QUI$_DISPLAY_FILE operation.
JBC$_NOMORECHAR	No more characteristics are defined, which indicates the termination of a QUI$_DISPLAY_CHARACTERISTIC wildcard operation.
JBC$_NOMOREENT	There are no more job entries for the specified user or current user name, which indicates termination of a QUI$_DISPLAY_ENTRY wildcard operation.
JBC$_NOMOREFILE	No more files are associated with the current job context, which indicates the termination of a QUI$_DISPLAY_FILE wildcard operation for the current job context.

JBC$_NOMOREFORM	No more forms are defined, which indicates the termination of a QUI$_DISPLAY_FORM wildcard operation.
JBC$_NOMOREJOB	No more jobs are associated with the current queue context, which indicates the termination of a QUI$_DISPLAY_JOB wildcard operation for the current queue context.
JBC$_NOMOREQUE	No more queues are defined, which indicates the termination of a QUI$_DISPLAY_QUEUE wildcard operation.
JBC$_NOQUECTX	No queue context has been established for a QUI$_DISPLAY_JOB or QUI$_DISPLAY_FILE operation.
JBC$_NOSUCHCHAR	The specified characteristic does not exist.
JBC$_NOSUCHENT	There is no job with the specified entry number, or there is no job for the specified user or current username.
JBC$_NOSUCHFILE	The specified file does not exist.
JBC$_NOSUCHFORM	The specified form does not exist.
JBC$_NOSUCHJOB	The specified job does not exist.
JBC$_NOSUCHQUE	The specified queue does not exist.

EXAMPLES

The following FORTRAN program demonstrates how a batch job can obtain information about itself from the system job queue file by using the $GETQUIW system service. Use of the QUI$M_SEARCH_THIS_JOB option in the QUI$_SEARCH_FLAGS input item requires that the calling program run as a batch job; otherwise, the $GETQUIW service returns a JBC$_NOSUCHJOB error.

1

```
! Declare system service related symbols
INTEGER*4       SYS$GETQUIW,
2               LIB$MATCH_COND,
2               STATUS
INCLUDE         '($QUIDEF)'

! Define item list structure
STRUCTURE       /ITMLST/
  UNION
    MAP
      INTEGER*2 BUFLEN, ITMCOD
      INTEGER*4 BUFADR, RETADR
    END MAP
    MAP
      INTEGER*4 END_LIST
    END MAP
  END UNION
END STRUCTURE

! Define I/O status block structure
STRUCTURE       /IOSBLK/
INTEGER*4       STS, ZEROED
END STRUCTURE
```

```
! Declare $GETQUIW item list and I/O status block
RECORD /ITMLST/ GETQUI_LIST(4)
RECORD /IOSBLK/ IOSB

! Declare variables used in $GETQUIW item list
CHARACTER*31    QUEUE_NAME
INTEGER*2       QUEUE_NAME_LEN
INTEGER*4       SEARCH_FLAGS,
2               ENTRY_NUMBER

! Initialize item list
GETQUI_LIST(1).BUFLEN = 4
GETQUI_LIST(1).ITMCOD = QUI$_SEARCH_FLAGS
GETQUI_LIST(1).BUFADR = %LOC(SEARCH_FLAGS)
GETQUI_LIST(1).RETADR = 0
GETQUI_LIST(2).BUFLEN = 4
GETQUI_LIST(2).ITMCOD = QUI$_ENTRY_NUMBER
GETQUI_LIST(2).BUFADR = %LOC(ENTRY_NUMBER)
GETQUI_LIST(2).RETADR = 0
GETQUI_LIST(3).BUFLEN = 31
GETQUI_LIST(3).ITMCOD = QUI$_QUEUE_NAME
GETQUI_LIST(3).BUFADR = %LOC(QUEUE_NAME)
GETQUI_LIST(3).RETADR = %LOC(QUEUE_NAME_LEN)
GETQUI_LIST(4).END_LIST = 0

SEARCH_FLAGS = QUI$V_SEARCH_THIS_JOB

! Call $GETQUIW service to obtain job information
STATUS = SYS$GETQUIW (,
2               %VAL(QUI$_DISPLAY_JOB),,
2               GETQUI_LIST,
2               IOSB,,)
IF (LIB$MATCH_COND (IOSB.STS, %LOC(JBC$_NOSUCHJOB))) THEN
    ! The search_this_job option can be used only by
    ! a batch job to obtain information about itself
    TYPE *, '<<< this job is not being run in batch mode>>>'
ENDIF
IF (STATUS) STATUS = IOSB.STS
IF (STATUS) THEN
    ! Display information
    TYPE *, 'Job entry number = ', ENTRY_NUMBER
    TYPE *, 'Queue name = ', QUEUE_NAME(1:QUEUE_NAME_LEN)
ELSE
    ! Signal error condition
    CALL LIB$SIGNAL (%VAL(STATUS))
ENDIF
END
```

The following FORTRAN program demonstrates how any job can obtain information about other jobs from the system job queue file by using the $GETQUIW system service. This program lists all print jobs in output queues with a job size of 500 blocks or more. It also displays queue name, job size, user name, and job name information for each job listed.

2

```
! Declare system service related symbols
INTEGER*4       SYS$GETQUIW,
2               STATUS_Q,
2               STATUS_J,
2               NOACCESS
INCLUDE         '($QUIDEF)'

! Define item list structure
STRUCTURE       /ITMLST/
  UNION
    MAP
      INTEGER*2 BUFLEN, ITMCOD
      INTEGER*4 BUFADR, RETADR
    END MAP
    MAP
      INTEGER*4 END_LIST
    END MAP
  END UNION
END STRUCTURE

! Define I/O status block structure
STRUCTURE       /IOSBLK/
INTEGER*4       STS, ZEROED
END STRUCTURE

! Declare $GETQUIW item lists and I/O status block
RECORD /ITMLST/ QUEUE_LIST(4)
RECORD /ITMLST/ JOB_LIST(6)
RECORD /IOSBLK/ IOSB

! Declare variables used in $GETQUIW item lists
CHARACTER*31    SEARCH_NAME
CHARACTER*31    QUEUE_NAME
CHARACTER*39    JOB_NAME
CHARACTER*12    USERNAME
INTEGER*2       SEARCH_NAME_LEN,
2               QUEUE_NAME_LEN,
2               JOB_NAME_LEN,
2               USERNAME_LEN
INTEGER*4       SEARCH_FLAGS,
2               JOB_SIZE,
2               JOB_STATUS

! Solicit queue name to search; it may be a wildcard name
TYPE 9000
ACCEPT 9010, SEARCH_NAME_LEN, SEARCH_NAME

! Initialize item list for the display queue operation
QUEUE_LIST(1).BUFLEN =  SEARCH_NAME_LEN
QUEUE_LIST(1).ITMCOD =  QUI$_SEARCH_NAME
QUEUE_LIST(1).BUFADR =  %LOC(SEARCH_NAME)
QUEUE_LIST(1).RETADR =  0
QUEUE_LIST(2).BUFLEN =  4
QUEUE_LIST(2).ITMCOD =  QUI$_SEARCH_FLAGS
QUEUE_LIST(2).BUFADR =  %LOC(SEARCH_FLAGS)
QUEUE_LIST(2).RETADR =  0
QUEUE_LIST(3).BUFLEN =  31
QUEUE_LIST(3).ITMCOD =  QUI$_QUEUE_NAME
QUEUE_LIST(3).BUFADR =  %LOC(QUEUE_NAME)
QUEUE_LIST(3).RETADR =  %LOC(QUEUE_NAME_LEN)
QUEUE_LIST(4).END_LIST = 0
```

```
! Initialize item list for the display job operation
JOB_LIST(1).BUFLEN =    4
JOB_LIST(1).ITMCOD =    QUI$_SEARCH_FLAGS
JOB_LIST(1).BUFADR =    %LOC(SEARCH_FLAGS)
JOB_LIST(1).RETADR =    0
JOB_LIST(2).BUFLEN =    4
JOB_LIST(2).ITMCOD =    QUI$_JOB_SIZE
JOB_LIST(2).BUFADR =    %LOC(JOB_SIZE)
JOB_LIST(2).RETADR =    0
JOB_LIST(3).BUFLEN =    39
JOB_LIST(3).ITMCOD =    QUI$_JOB_NAME
JOB_LIST(3).BUFADR =    %LOC(JOB_NAME)
JOB_LIST(3).RETADR =    %LOC(JOB_NAME_LEN)
JOB_LIST(4).BUFLEN =    12
JOB_LIST(4).ITMCOD =    QUI$_USERNAME
JOB_LIST(4).BUFADR =    %LOC(USERNAME)
JOB_LIST(4).RETADR =    %LOC(USERNAME_LEN)
JOB_LIST(5).BUFLEN =    4
JOB_LIST(5).ITMCOD =    QUI$_JOB_STATUS
JOB_LIST(5).BUFADR =    %LOC(JOB_STATUS)
JOB_LIST(5).RETADR =    0
JOB_LIST(6).END_LIST =  0

! Request search of all jobs present in output queues; also force
! wildcard mode to maintain the internal search context block after
! the first call when a non-wild queue name is entered--this preserves
! queue context for the subsequent display job operation
SEARCH_FLAGS = (QUI$M_SEARCH_WILDCARD .OR.
2               QUI$M_SEARCH_SYMBIONT .OR.
2               QUI$M_SEARCH_ALL_JOBS)

! Dissolve any internal search context block for the process
STATUS_Q = SYS$GETQUIW (,%VAL(QUI$_CANCEL_OPERATION),,,,,)

! Locate next output queue; loop until an error status is returned
DO WHILE (STATUS_Q)
    STATUS_Q = SYS$GETQUIW (,
2                       %VAL(QUI$_DISPLAY_QUEUE),,
2                       QUEUE_LIST,
2                       IOSB,,)
    IF (STATUS_Q) STATUS_Q = IOSB.STS
    IF (STATUS_Q) TYPE 9020, QUEUE_NAME(1:QUEUE_NAME_LEN)
    STATUS_J = 1

    ! Get information on next job in queue; loop until error return
    DO WHILE (STATUS_Q .AND. STATUS_J)
        STATUS_J = SYS$GETQUIW (,
2                       %VAL(QUI$_DISPLAY_JOB),,
2                       JOB_LIST,
2                       IOSB,,)
        IF (STATUS_J) STATUS_J = IOSB.STS
        IF ((STATUS_J) .AND. (JOB_SIZE .GE. 500)) THEN
            NOACCESS = (JOB_STATUS .AND. QUI$M_JOB_INACCESSIBLE)
            IF (NOACCESS .NE. 0) THEN
                TYPE 9030, JOB_SIZE
            ELSE
                TYPE 9040, JOB_SIZE,
2                   USERNAME(1:USERNAME_LEN),
2                   JOB_NAME(1:JOB_NAME_LEN)
            ENDIF
        ENDIF
    ENDDO
ENDDO
```

```
9000    FORMAT (' Enter queue name to search: ', $)
9010    FORMAT (Q, A31)
9020    FORMAT ('0Queue name = ', A)
9030    FORMAT ('   Job size = ', I5, '   <no read access privilege>')
9040    FORMAT ('   Job size = ', I5,
        2       '   Username = ', A, T46,
        2       '   Job name = ', A)
        END
```

$GETQUIW Get Queue Information and Wait for Completion

The Get Queue Information and Wait for Completion service returns information about queues and jobs initiated from those queues. The $SNDJBC service is the major interface to the VMS Job Controller, which is the VMS queue and accounting manager. For a discussion of the different types of job and queue, see the DESCRIPTION section of the $SNDJBC service.

The $GETQUIW service completes synchronously; that is, it returns to the caller with the requested information. For asynchronous completion, you use the Get Queue Information ($GETQUI) service; $GETQUI returns to the caller after queuing the information request, without waiting for the information to be returned.

In all other respects, $GETQUIW is identical to $GETQUI. For all other information about the $GETQUIW service, refer to the documentation of $GETQUI.

For additional information about system service completion, refer to the Synchronize ($SYNCH) service.

FORMAT

SYS$GETQUIW *[efn] ,func [,nullarg] [,itmlst] [,iosb] [,astadr] [,astprm]*

$GETSYI Get Systemwide Information

The Get Systemwide Information service returns information about the local VAX system or about other VAX systems in a cluster.

For Version 5.0 of VMS, both the $GETSYI service and the Get Systemwide Information and Wait ($GETSYIW) services complete synchronously; that is, they return to the caller with the requested information.

However, in the future, $GETSYI may be modified to complete asynchronously; that is, it will return to the caller after queueing the information request, without waiting for the information to be returned. For this reason, DIGITAL recommends that you use the $GETSYIW service for synchronous completion.

For additional information about system service completion, refer to the Synchronize ($SYNCH) service and to the *Introduction to VMS System Services*.

FORMAT

SYS$GETSYI *[efn] ,[csidadr] ,[nodename] ,itmlst [,iosb] [,astadr] [,astprm]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

efn

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

Number of the event flag to be set when the $GETSYI request completes. The **efn** argument is a longword containing this number; however, $GETSYI uses only the low-order byte.

Upon request initiation, $GETSYI clears the specified event flag (or event flag 0 if **efn** was not specified). Then, when the request completes, the specified event flag (or event flag 0) is set.

csidadr

VMS usage: **process_id**
type: **longword (unsigned)**
access: **modify**
mechanism: **by reference**

Cluster system identification of the VAX node about which $GETSYI is to return information. The **csidadr** argument is the address of a longword containing this identification value.

The cluster-connection software assigns the cluster system identification of a VAX node. You may obtain this information by using the DCL command SHOW CLUSTER. The value of the cluster system identification for a VAX node is not permanent; a new value is assigned to a VAX node whenever it joins or rejoins the VAXcluster.

You may also specify a VAX node to $GETSYI by using the **nodename** argument. If you specify **csidadr**, you need not specify **nodename**, and vice versa. If you specify both, they must identify the same VAX node. If you specify neither, $GETSYI returns information about the local VAX node. However, for wildcard operations, you must use the **csidadr** argument.

If you specify **csidadr** as –1, $GETSYI assumes a wildcard operation and returns the requested information for each VAX node in the cluster, one node per call. In this case, the program should test for the condition value SS$_NOMORENODE after each call to $GETSYI and should stop calling $GETSYI when SS$_NOMORENODE is returned.

nodename

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

Name of the VAX node about which $GETSYI is to return information. The **nodename** argument is the address of a character string descriptor pointing to this name string.

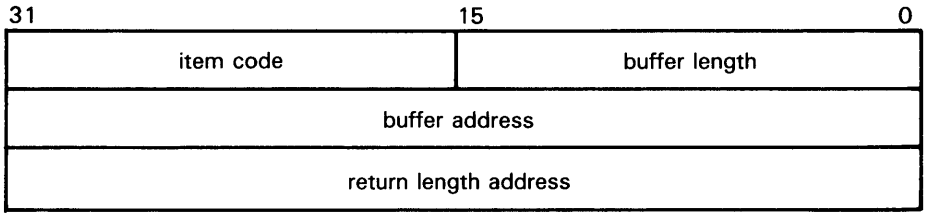
The node name string must contain from 1 to 15 characters and must correspond exactly to the VAX node name; no trailing blanks or abbreviations are permitted.

You may also specify a VAX node to $GETSYI by using the **csidadr** argument. See the description of **csidadr**.

itmlst

VMS usage: **item_list_3**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Item list specifying which information is to be returned about the VAX node or nodes. The **itmlst** argument is the address of a list of item descriptors, each of which describes an item of information. The list of item descriptors is terminated by a longword of 0. The following diagram depicts a single item descriptor.



ZK-1705-84

$GETSYI Item Descriptor Fields

buffer length

A word containing a user-supplied integer specifying the length (in bytes) of the buffer in which $GETSYI is to write the information. The length of the buffer needed depends upon the item code specified in the **item code** field of the item descriptor. If the value of **buffer length** is too small, $GETSYI truncates the data.

item code

A word containing a user-supplied symbolic code specifying the item of information that $GETSYI is to return. The $SYIDEF macro defines these codes. A description of each item code is given in the $GETSYI Item Codes section.

buffer address

A longword containing the user-supplied address of the buffer in which $GETSYI is to write the information.

return length address

A longword containing the user-supplied address of a word in which $GETSYI writes the length in bytes of the information it actually returned.

$GETSYI Item Codes

SYI$_ACTIVECPU_CNT

When you specify SYI$_ACTIVECPU_CNT, $GETSYI returns a count of CPUs actively participating in the current boot of the Symmetric MultiProcessing (SMP) system.

Because this number is a longword, the **buffer length** field in the item descriptor should specify 4 (bytes).

SYI$_AVAILCPU_CNT

When you specify SYI$_AVAILCPU_CNT, $GETSYI returns the number of CPUs available in the current boot of the SMP system.

Because this number is a longword, the **buffer length** field in the item descriptor should specify 4 (bytes).

SYI$_BOOTTIME

When you specify SYI$_BOOTTIME, $GETSYI returns the time when the VAX node was booted. The $GETSYI service returns this information only for the local VAX node.

Because the returned time is in the standard 64-bit absolute time format, the **buffer length** field in the item descriptor should specify 8 (bytes).

SYI$_CHARACTER_EMULATED

When you specify SYI$_CHARACTER_EMULATED, $GETSYI returns the number 1 if the character string instructions are emulated on the CPU and a 0 if they are not. The $GETSYI service returns this information only for the local VAX node.

Because this number is a Boolean value (1 or 0), the **buffer length** field in the item descriptor should specify 1 (byte).

SYI$_CLUSTER_EVOTES

When you specify SYI$_CLUSTER_EVOTES, $GETSYI returns the number of votes expected to be found in the VAXcluster. The VAXcluster determines this value by selecting the highest number from all of the following: each node's SYSGEN parameter EXPECTED_VOTES, the sum of the votes currently in the VAXcluster, and the previous value for the number of expected votes.

Because this number is a word in length, the **buffer length** field in the item descriptor should specify 2 (bytes).

SYI$_CLUSTER_FSYSID

When you specify SYI$_CLUSTER_FSYSID, $GETSYI returns the system identification of the founding node, which is the first node in the cluster to boot.

The cluster management software assigns this system identification to the node. You may obtain this information by using the DCL command SHOW CLUSTER. Because the system identification is a 6-byte hexadecimal numbers, the **buffer length** field in the item descriptor should specify 6 (bytes).

SYI$_CLUSTER_FTIME

When you specify SYI$_CLUSTER_FTIME, $GETSYI returns the time when the founding node is booted. The founding node is the first node in the cluster to boot.

Because the returned time is in the standard 64-bit absolute time format, the **buffer length** field in the item descriptor should specify 8 (bytes).

SYI$_CLUSTER_MEMBER

When you specify SYI$_CLUSTER_MEMBER, $GETSYI returns the membership status of the node in the cluster. The membership status specifies whether the node is currently a member of the cluster.

Because the membership status of a node is described in a 1-byte bit field, the **buffer length** field in the item descriptor should specify 1 (byte). If bit 0 in the bit field is set, the node is a member of the cluster; if it is clear, then it is not a member of the cluster.

SYI$_CLUSTER_NODES

When you specify SYI$_CLUSTER_NODES, $GETSYI returns the number (in decimal) of nodes currently in the cluster.

Because this number is a word in length, the **buffer length** field in the item descriptor should specify 2 (bytes).

SYI$_CLUSTER_QUORUM

When you specify SYI$_CLUSTER_QUORUM, $GETSYI returns the number (in decimal) that is the total of the quorum values held by all nodes in the cluster. Each node's quorum value is derived from its SYSGEN parameter EXPECTED_VOTES.

Because this number is a word in length, the **buffer length** field in the item descriptor should specify 2 (bytes).

SYI$_CLUSTER_VOTES

When you specify SYI$_CLUSTER_VOTES, $GETSYI returns the total number of votes held by all nodes in the cluster. The number of votes held by any one node is determined by that node's SYSGEN parameter VOTES.

Because this decimal number is a word in length, the **buffer length** field in the item descriptor should specify 2 (bytes).

SYI$_CONTIG_GBLPAGES

When you specify SYI$_CONTIG_GBLPAGES, $GETSYI returns the maximum number of free, contiguous global pages. This number is the largest size global section that can be created.

Because this number is a longword, the **buffer length** in the item descriptor should specify 4 (bytes).

SYI$_CPU

When you specify SYI$_CPU, $GETSYI returns the CPU processor type of the node. The $GETSYI service returns this information only for the local VAX node.

Because the processor type is a longword decimal number, the **buffer length** field in the item descriptor should specify 4 (bytes).

The $PRDEF macro defines the following symbols for the processor types.

Processor	Symbol
VAX–11 730	PR$_SID_TYP730
VAX–11 780, 785	PR$_SID_TYP780
VAX–11 750	PR$_SID_TYP750
MicroVAX I	PR$_SID_TYPUV1
MicroVAX II series	PR$_SID_TYPUV2
VAXstation 2000	PR$_SID_TYP410
VAX 8600, 8650	PR$_SID_TYP790
VAX 8200, 8300, 8250, 8350	PR$_SID_TYP8SS
VAX 8530, 8550, 8700, 8800	PR$_SID_TYP8NN
MicroVAX 3000 Series VAXstation 3000 Series	PR$_SID_TYP650

For information about extended processor type codes, see the description for the SYI$_XCPU item code.

SYI$_DECIMAL_EMULATED

When you specify SYI$_DECIMAL_EMULATED, $GETSYI returns the number 1 if the decimal string instructions are emulated on the CPU and a 0 if they are not. The $GETSYI service returns this information only for the local VAX node.

Because this number is a Boolean value (1 or 0), the **buffer length** field in the item descriptor should specify 1 (byte).

SYI$_D_FLOAT_EMULATED

When you specify SYI$_D_FLOAT_EMULATED, $GETSYI returns the number 1 if the D_floating instructions are emulated on the CPU, and 0 if they are not. The $GETSYI service returns this information only for the local VAX node.

Because this number is a Boolean value (1 or 0), the **buffer length** field in the item descriptor should specify 1 (byte).

SYI$_ERRORLOGBUFFERS

When you specify SYI$_ERRORLOGBUFFERS, $GETSYI returns the number of system pages in use as buffers for the Error Logger.

Because this number is a word in length, the **buffer length** field in the item descriptor should specify 2 (bytes).

SYI$_F_FLOAT_EMULATED

When you specify SYI$_F_FLOAT_EMULATED, $GETSYI returns the number 1 if the F_floating instructions are emulated on the CPU, and 0 if they are not. The $GETSYI service returns this information only for the local VAX node.

Because this number is a Boolean value (1 or 0), the **buffer length** field in the item descriptor should specify 1 (byte).

SYI$_FREE_GBLPAGES

When you specify SYI$_FREE_GBLPAGES, $GETSYI returns the current number of free global pages. The SYSGEN parameter GBLPAGES sets the number of global pages that can exist systemwide.

Because the current number is a longword, the **buffer length** in the item descriptor should specify 4 (bytes).

SYI$_FREE_GBLSECTS

When you specify SYI$_FREE_GBLSECTS, $GETSYI returns the current number of free global section table entries. The SYSGEN parameter GBLSECTIONS sets the maximum number of global sections that can exist systemwide.

Because the current number is a longword, the **buffer length** in the item descriptor should specify 4 (bytes).

SYI$_G_FLOAT_EMULATED

When you specify SYI$_G_FLOAT_EMULATED, $GETSYI returns the number 1 if the G_floating instructions are emulated on the CPU, and a 0 if they are not. The $GETSYI service returns this information only for the local VAX node.

Because this number is a Boolean value (1 or 0), the **buffer length** field in the item descriptor should specify 1 (byte).

SYI$_H_FLOAT_EMULATED

When you specify SYI$_H_FLOAT_EMULATED, $GETSYI returns the number 1 if the H_floating instructions are emulated on the CPU, and a 0 if they are not. The $GETSYI service returns this information only for the local VAX node.

Because this number is a Boolean value (1 or 0), the **buffer length** field in the item descriptor should specify 1 (byte).

SYI$_HW_MODEL

When you specify SYI$_HW_MODEL, $GETSYI returns a small integer that can be used to identify the VAX model type of the node. The $VAXDEF macro in SYS$LIBRARY:STARLET defines the model type integers. See the table under SYI$_HW_NAME for the VAX model processor names and the corresponding model types.

Because the HW_MODEL is a word, the buffer length field in the item descriptor should specify 2 (bytes).

SYI$_HW_NAME

When you specify SYI$_HW_NAME, $GETSYI returns the VAX model name string of the node. The VAX model name is a character string that describes the model of the VAX node (such as VAX 8800, MicroVAX II). The VAX model name usually corresponds to the nameplate that appears on the outside of the CPU cabinet. This item code supersedes SYI$_NODE_HWTYPE, which is supported in this release for compatibility with VAX/VMS Version 4.n. DIGITAL recommends that you use SYI$_HW_NAME. You should update old programs with the new item code, as convenient.

Because the HW_NAME can include up to 31 characters, the buffer length field in the item descriptor should specify 31 (bytes).

The following table lists the VAX model processor names and the corresponding model types.

VAX Model Processor Name	VAX Model Type
VAX-11/730	VAX$K_V730
VAX-11/750	VAX$K_V750
VAX-11/780	VAX$K_V780
VAX-11/785	VAX$K_V785
MicroVAX I	VAX$K_VUV1
VAXstation I	VAX$K_VWS1
MicroVAX II	VAX$K_VUV2
VAXstation II	VAX$K_VWS2
VAXstation II/GPX	VAX$K_VWSD
VAX 8200	VAX$K_V8200
VAX 8250	VAX$K_V8250
VAX 8300	VAX$K_V8300
VAX 8350	VAX$K_V8350
VAX 8530	VAX$K_V8500

VAX Model Processor Name	VAX Model Type
VAX 8550	VAX$K_V8550
VAX 8600	VAX$K_V8600
VAX 8650	VAX$K_V8650
VAX 8700	VAX$K_V8700
VAX 8800	VAX$K_V8800
VAXstation 2000	VAX$K_VWS2000
MicroVAX 2000	VAX$K_VUV2000
VAXstation 2000	VAX$K_VWSD2000
MicroVAX 3000 Series	VAX$K_V650
VAXstation 3000 Series	VAX$K_V65W
VAXstation 3000 Series	VAX$K_V65D

SYI$_NODE_AREA

When you specify SYI$_NODE_AREA, $GETSYI returns the DECNET area of the node.

Because the DECNET area is a longword decimal number, the **buffer length** field in the item descriptor should specify 4 (bytes).

SYI$_NODE_CSID

When you specify SYI$_NODE_CSID, $GETSYI returns the cluster system ID (CSID) of the VAX node. The CSID is a longword hexadecimal number assigned to the node by the cluster management software.

Because the CSID is a longword, the **buffer length** field in the item descriptor should specify 4 (bytes).

SYI$_NODE_EVOTES

When you specify SYI$_NODE_EVOTES, $GETSYI returns the number of votes the node expects to find in the VAXcluster. This number is determined by the SYSGEN parameter EXPECTED_VOTES.

Because the number is a word in length, the **buffer length** field in the item descriptor should specify 2 (bytes).

SYI$_NODE_HWVERS

When you specify SYI$_NODE_HWVERS, $GETSYI returns the hardware version of the node. The high word of the **buffer length** contains the VAX CPU type. The $VAXDEF macro defines the VAX CPU type.

Because the hardware version is a 12-byte hexadecimal number, the **buffer length** field in the item descriptor should specify 12 (bytes).

SYI$_NODE_NUMBER

When you specify SYI$_NODE_NUMBER, $GETSYI returns the DECNET number of the node.

Because the DECNET number is a longword decimal number, the **buffer length** field in the item descriptor should specify 4 (bytes).

SYI$_NODE_QUORUM

When you specify SYI$_NODE_QUORUM, $GETSYI returns the value (in decimal) of the quorum held by the node. This number is derived from the node's SYSGEN parameter EXPECTED_VOTES.

Because this number is a word in length, the **buffer length** field in the item descriptor should specify 2 (bytes).

SYI$_NODE_SWINCARN

When you specify SYI$_NODE_SWINCARN, $GETSYI returns the software incarnation of the node.

Because the software incarnation of the node is an 8-byte hexadecimal number, the **buffer length** field in the item descriptor should specify 8 (bytes).

SYI$_NODE_SWTYPE

When you specify SYI$_NODE_SWTYPE, $GETSYI returns the software type of the node. The software type indicates whether the node is a VMS system or an HSC storage controller.

Because the software type is a 4-byte ASCII string, the **buffer length** field in the item descriptor should specify 4 (bytes).

SYI$_NODE_SWVERS

When you specify SYI$_NODE_SWVERS, $GETSYI returns the software version of the node.

Because the software version is a 4-byte ASCII string, the **buffer length** field in the item descriptor should specify 4 (bytes).

SYI$_NODE_SYSTEMID

When you specify SYI$_NODE_SYSTEMID, $GETSYI returns the system identification of the node.

The cluster management software assigns this system identification to the node. You may obtain this information by using the DCL command SHOW CLUSTER. Because the system identification is a 6-byte hexadecimal number, the **buffer length** field in the item descriptor should specify 6 (bytes).

SYI$_NODE_VOTES

When you specify SYI$_NODE_VOTES, $GETSYI returns the number (in decimal) of votes held by the node. This number is determined by the node's SYSGEN parameter VOTES.

Because this number is a word in length, the **buffer length** field in the item descriptor should specify 2 (bytes).

SYI$_NODENAME

When you specify SYI$_NODENAME, $GETSYI returns, as a character string, the name of the node in the returned length area specified in the item list.

Because this name can inculde up to 15 characters, the **buffer length** field in the item descriptor should specify 15 (bytes).

SYI$_PAGEFILE_FREE

When you specify SYI$_PAGEFILE_FREE, $GETSYI returns the number of free pages in the currently installed paging files. The $GETSYI service returns this information only for the local VAX node.

Because this number is a longword, the **buffer length** field in the item descriptor should specify 4 (bytes).

SYI$_PAGEFILE_PAGE

When you specify SYI$_PAGEFILE_PAGE, $GETSYI returns the number of pages in the currently installed paging files. The $GETSYI service returns this information only for the local VAX node.

Because this number is a longword, the **buffer length** field in the item descriptor should specify 4 (bytes).

SYI$_SCS_EXISTS

When you specify SYI$_SCS_EXISTS, $GETSYI returns a longword value that is interpreted as Boolean. If the value is 1, the System Communication Subsystem (SCS) is currently loaded on the VAX node; if the value is 0, the SCS is not currently loaded.

SYI$_SID

When you specify SYI$_SID, $GETSYI returns the contents of the system identification register of the VAX node. For more information about the meaning of the contents of the system identification register, see the *VAX Hardware Handbook*. The $GETSYI service returns this information only for the local VAX node.

Because the value of this register is a longword hexadecimal number, the **buffer length** field in the item descriptor should specify 4 (bytes).

SYI$_SWAPFILE_FREE

When you specify SYI$_SWAPFILE_FREE, $GETSYI returns the number of free pages in the currently installed swapping files. The $GETSYI service returns this information only for the local VAX node.

Because this number is a longword, the **buffer length** field in the item descriptor should specify 4 (bytes).

SYI$_SWAPFILE_PAGE

When you specify SYI$_SWAPFILE_PAGE, $GETSYI returns the number of pages in the currently installed swapping files. The $GETSYI service returns this information only for the local VAX node.

Because this number is a longword, the **buffer length** field in the item descriptor should specify 4 (bytes).

SYI$_VERSION

When you specify SYI$_VERSION, $GETSYI returns, as a charcter string, the software version number of the VMS operating system running on the VAX node. The $GETSYI service returns this information only for the local VAX node.

Because the version number is 8-byte blank-filled, the **buffer length** field in the item descriptor should specify 8 (bytes).

SYI$_XCPU

When you specify SYI$_XCPU, $GETSYI returns the extended CPU processor type of the node. The $GETSYI service returns this information only for the local VAX node.

You should obtain the general processor type value first by using the SYI$_CPU item code. For some of the general processor types, extended processor type information is provided by the item code, SYI$_XCPU. For other general processor types, the value returned by the SYI$_XCPU item code is currently undefined.

Because the processor type is a longword decimal number, the **buffer length** field in the item descriptor should specify 4 (bytes).

The $PRDEF macro defines the following symbols for the extended processor types.

VAX Processor Type Symbol	Extended Processor Type	Extended Processor Symbol
PR$_SID_TYPUV	MicroVAX II VAXstation II	PR$_XSID_UV_UV2
	MicroVAX 2000 VAXstation 2000	PR$_XSID_UV_410
PR$_SID_TYPCV	MicroVAX 3000 Series VAXstation 3000 Series	PR$_XSID_CV_650
PR$_SID_TYP8NN	VAX 8530	PRS$_XSID_N8500
	VAX 8550	PRS$_XSID_N8550
	VAX 8700	PRS$_XSID_N8700
	VAX 8800	PRS$_XSID_N8800

SYI$_XSID

When you specify SYI$_XSID, $GETSYI returns processor-specific information. For the MicroVAX II, this information is the contents of the system type register of the VAX node. The system type register contains the full extended information used in determining the extended system type codes. For other processors, the data returned by SYI$_XSID are currently undefined.

Because the value of this register is a longword hexadecimal number, the **buffer length** field in the item descriptor should specify 4 (bytes).

SYI$_xxxx

When you specify SYI$_xxxx, $GETSYI returns the current value of the SYSGEN parameter named *xxxx* for the VAX node. The $GETSYI service returns this information only for the local VAX node.

The buffer must specify a longword into which $GETSYI writes the value of the specified SYSGEN parameter. For a list and description of all system parameters, refer to the *VMS System Generation Utility Manual.*

iosb

VMS usage: **io_status_block**
type: **quadword (unsigned)**
access: **write only**
mechanism: **by reference**

I/O status block to receive the final completion status. The **iosb** argument is the address of the quadword I/O status block.

When you specify the **iosb** argument, $GETSYI sets the quadword to zero upon request initiation. Upon request completion, a condition value is returned to the first longword; the second longword is reserved to DIGITAL.

Though this argument is optional, DIGITAL strongly recommends that you specify it, for the following reasons:

- If you are using an event flag to signal the completion of the service, you can test the I/O status block for a condition value to be sure that the event flag was not set by an event other than service completion.
- If you are using the $SYNCH service to synchronize completion of the service, the I/O status block is a required argument for $SYNCH.
- The condition value returned in R0 and the condition value returned in the I/O status block provide information about different aspects of the call to the $GETSYI service. The condition value returned in R0 gives you information about the success or failure of the service call itself; the condition value returned in the I/O status block gives you information about the success or failure of the service operation. Therefore, to accurately assess the success or failure of the call to $GETSYI, you must check the condition values returned in both R0 and the I/O status block.

astadr

VMS usage: **ast_procedure**
type: **procedure entry mask**
access: **call without stack unwinding**
mechanism: **by reference**

AST service routine to be executed when $GETSYI completes. The **astadr** argument is the address of the entry mask of this routine.

If you specify **astadr**, the AST routine executes at the same access mode as the caller of the $GETSYI service.

astprm

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

AST parameter to be passed to the AST service routine specified by the **astadr** argument. The **astprm** argument is the longword parameter.

DESCRIPTION

This service uses the process's AST limit quota (ASTLM).

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_NOMORENODE	You requested a wildcard operation, and $GETSYI has returned information about all available VAX nodes.
SS$_BADPARAM	The item list contains an invalid item code.
SS$_ACCVIO	The caller cannot read the item list; cannot write to the buffer specified by the **buffer address** field in an item descriptor; or cannot write to the **return length address** field in an item descriptor.
SS$_EXASTLM	The process has exceeded its AST limit quota.
SS$_NOSUCHNODE	The specified VAX node does not exist or is not currently a member of the VAXcluster.

CONDITION VALUES RETURNED IN THE I/O STATUS BLOCK

Same as those returned in R0.

EXAMPLE

The following FORTRAN program demonstrates how to use the $GETSYIW service to obtain the operating system version number string and the system's node name.

```
! Declare system service related symbols
INTEGER*4       SYS$GETSYIW,
2               STATUS
! External declaration is an alternative to including $SYIDEF
EXTERNAL        SYI$_VERSION,
2               SYI$_NODENAME

! Define item list structure
STRUCTURE       /ITMLST/
  UNION
    MAP
      INTEGER*2 BUFLEN
      INTEGER*2 ITMCOD
      INTEGER*4 BUFADR
      INTEGER*4 RETADR
    END MAP
    MAP
      INTEGER*4 END_LIST
    END MAP
  END UNION
END STRUCTURE

! Define I/O status block structure
STRUCTURE       /IOSBLK/
INTEGER*4       STS, RESERVED
END STRUCTURE
```

```
! Declare $GETSYIW item list and I/O status block
RECORD /ITMLST/ GETSYI_LIST(3)
RECORD /IOSBLK/ IOSB

! Declare variables used in $GETSYIW item list
CHARACTER*8     VERSION
CHARACTER*15    NODENAME
INTEGER*2       VERSION_LEN,
2               NODENAME_LEN

! Initialize item list
GETSYI_LIST(1).BUFLEN = 8
GETSYI_LIST(1).ITMCOD = %LOC(SYI$_VERSION)
GETSYI_LIST(1).BUFADR = %LOC(VERSION)
GETSYI_LIST(1).RETADR = %LOC(VERSION_LEN)
GETSYI_LIST(2).BUFLEN = 15
GETSYI_LIST(2).ITMCOD = %LOC(SYI$_NODENAME)
GETSYI_LIST(2).BUFADR = %LOC(NODENAME)
GETSYI_LIST(2).RETADR = %LOC(NODENAME_LEN)
GETSYI_LIST(3).END_LIST = 0

! Display the system version number string
STATUS = SYS$GETSYIW (,,,GETSYI_LIST,IOSB,,)
IF (STATUS) STATUS = IOSB.STS
IF (.NOT. STATUS) CALL LIB$SIGNAL (%VAL(STATUS))

TYPE *, 'System version is ', VERSION(1:VERSION_LEN)
END
```

$GETSYIW Get Systemwide Information and Wait

The Get Systemwide Information and Wait service returns information about the local VAX system or about other VAX systems in a cluster.

For Version 5.0 of VMS, the $GETSYIW service is identical to the Get Systemwide Information ($GETSYI) service. Both services return the same information, and both complete synchronously; that is, they return to the caller with the requested information.

In the future, however, the $GETSYI service may be modified to complete asynchronously; that is, it will return to the caller after queueing the information request, without waiting for the information to be returned. For this reason, DIGITAL recommends that you use the $GETSYIW service for synchronous completion.

For all other information about the $GETSYIW service, refer to the documentation of $GETSYI.

For additional information about system service completion, refer to the Synchronize ($SYNCH) service and to the *Introduction to VMS System Services*.

FORMAT

SYS$GETSYIW *[efn],[csidadr],[nodename],itmlst [,iosb][,astadr][,astprm]*

You must specify either the **csidadr** or the **nodename** argument, but not both. For wildcard operations, however, you must use the **csidadr** argument.

$GETTIM Get Time

The Get Time service returns the current system time in 64-bit format.

FORMAT

SYS$GETTIM *timadr*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENT

timadr
VMS usage: **date_time**
type: **quadword (unsigned)**
access: **write only**
mechanism: **by reference**

Address of a quadword to receive the current time in 64-bit format.

DESCRIPTION

The system time is updated every 10 milliseconds, and the time is returned in 100-nanosecond units from the system base time.

For additional information about the system time, see the *Introduction to VMS System Services.*

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The quadword to receive the time cannot be written by the caller.

$GETUAI Get User Authorization Information

The Get User Authorization Information service returns authorization information about a specified user.

FORMAT

SYS$GETUAI *[nullarg] ,[nullarg] ,usrnam ,itmlst ,[nullarg] ,[nullarg] ,[nullarg]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

nullarg

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

Place-holding argument reserved by DIGITAL.

usrnam

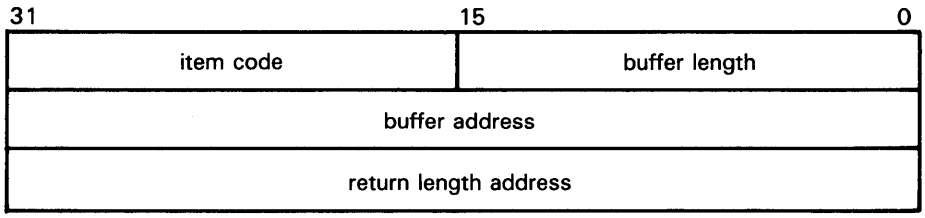
VMS usage: **char_string**
type: **character-coded text string**
access: **read only**
mechanism: **by descriptor—fixed-length string descriptor**

Name of the user about whom $GETUAI returns authorization information. The **usrnam** argument is the address of a descriptor pointing to a character text string containing the user name. The user name string may contain a maximum of 12 alphanumeric characters.

itmlst

VMS usage: **item_list_3**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Item list specifying which information from the specified user's user authorization file (UAF) record is to be returned. The **itmlst** argument is the address of a list of one or more item descriptors, each of which specifies an item code. The item list is terminated by an item code of 0 or by a longword of 0. The following diagram depicts the structure of a single item descriptor.



ZK-1705-84

$GETUAI Item Descriptor Fields

buffer length

A word specifying the length (in bytes) of the buffer in which $GETUAI is to write the information. The length of the buffer varies depending on the item code specified in the **item code** field of the item descriptor and is given in the description of each item code. If the value of **buffer length** is too small, $GETUAI truncates the data.

item code

A word containing a user-supplied symbolic code specifying the item of information that $GETUAI is to return. The $UAIDEF macro defines these codes, which have the following format:

UAI$_code

Each item code is described under $GETUAI Item Codes.

buffer address

A longword containing the user-supplied address of the buffer in which $GETUAI is to write the information.

return length address

A longword containing the user-supplied address of a word in which $GETUAI writes the length in bytes of the information it actually returned.

$GETUAI Item Codes

UAI$_ACCOUNT

When you specify UAI$_ACCOUNT, $GETUAI returns, as a blank-filled character string, the account name of the user.

Because an account name can include up to 8 characters plus a size-byte prefix, the buffer length field of the item descriptor should specify 9 (bytes).

UAI$_ASTLM

When you specify UAI$_ASTLM, $GETUAI returns the AST queue limit.

Because this decimal number is a word in length, the buffer length field in the item descriptor should specify 2 (bytes).

UAI$_BATCH_ACCESS_P

When you specify UAI$_BATCH_ACCESS_P, $GETUAI returns, as a 3-byte value, the range of times during which batch access is permitted for primary days. Each bit set represents a 1-hour period, from bit 0 as midnight to 1 a.m. to bit 23 as 11 p.m. to midnight.

The buffer length field in the item descriptor should specify 3 (bytes).

UAI$_BATCH_ACCESS_S

When you specify UAI$_BATCH_ACCESS_S, $GETUAI returns, as a 3-byte value, the range of times during which batch access is permitted for secondary days. Each bit set represents a one-hour period, from bit 0 as midnight to 1 a.m. to bit 23 as 11 p.m. to midnight.

The buffer length field in the item descriptor should specify 3 (bytes).

UAI$_BIOLM

When you specify UAI$_BIOLM, $GETUAI returns the buffered I/O count.

Because this decimal number is a word in length, the buffer length field in the item descriptor should specify 2 (bytes).

UAI$_BYTLM

When you specify UAI$_BYTLM, $GETUAI returns the buffered I/O byte limit.

Because the buffered I/O byte limit is a longword decimal number, the buffer length field in the item descriptor should specify 4 (bytes).

UAI$_CLITABLES

When you specify UAI$_CLITABLES, $GETUAI returns, as a character string, the name of the user-defined CLI table for the account, if any.

Because the CLI table name can include up to 31 characters plus a size-byte prefix, the buffer length field of the item descriptor should specify 32 (bytes).

UAI$_CPUTIM

When you specify UAI$_CPUTIM, $GETUAI returns the maximum CPU time limit (per session) for the process in 10-millisecond units.

Because the maximum CPU time limit is a longword decimal number, the buffer length field in the item descriptor should specify 4 (bytes).

UAI$_DEFCLI

When you specify UAI$_DEFCLI, $GETUAI returns, as an RMS file name component, the name of the command language interpreter used to execute the specified batch job. The file specification returned assumes the device name and directory SYS$SYSTEM and the file type EXE.

Because a file name can include up to 31 characters plus a size-byte prefix, the buffer length field in the item descriptor should specify 32 (bytes).

UAI$_DEFDEV

When you specify UAI$_DEFDEV, $GETUAI returns, as a 1- to 31-character string, the name of the default device.

Because the device name string can include up to 31 characters plus a size-byte prefix, the buffer length field in the item descriptor should specify 32 (bytes).

UAI$_DEFDIR

When you specify UAI$_DEFDIR, $GETUAI returns, as a 1- to 63-character string, the name of the default directory.

Because the directory name string can include up to 63 characters plus a size-byte prefix, the buffer length field in the item descriptor should specify 64 (bytes).

UAI$_DEF_PRIV

When you specify UAI$_DEF_PRIV, $GETUAI returns the default privileges for the user.

Because the default privileges are returned as a quadword value, the buffer length field in the item descriptor should specify 8 (bytes).

UAI$_DFWSCNT

When you specify UAI$_DFWSCNT, $GETUAI returns the default working set size.

Because the default working set size is a longword decimal number, the buffer length field in the item descriptor should specify 4 (bytes).

UAI$_DIOLM

When you specify UAI$_DIOLM, $GETUAI returns the direct I/O count limit.

Because this decimal number is a word in length, the buffer length field in the item descriptor should specify 2 (bytes).

UAI$_DIALUP_ACCESS_P

When you specify UAI$_DIALUP_ACCESS_P, $GETUAI returns, as a 3-byte value, the range of times during which dialup access is permitted for primary days. Each bit set represents a 1-hour period, from bit 0 as midnight to 1 a.m. to bit 23 as 11 p.m. to midnight.

The buffer length field in the item descriptor should specify 3 (bytes).

UAI$_DIALUP_ACCESS_S

When you specify UAI$_DIALUP_ACCESS_S, $GETUAI returns, as a 3-byte value, the range of times during which dialup access is permitted for secondary days. Each bit set represents a 1-hour period, from bit 0 as midnight to 1 a.m. to bit 23 as 11 p.m. to midnight.

The buffer length field in the item descriptor should specify 3 (bytes).

UAI$_ENCRYPT

When you specify UAI$_ENCRYPT, $GETUAI returns a code indicating the encryption algorithm for the primary password.

Because the encryption algorithm is a byte in length, the buffer length field in the item descriptor should specify 1 (byte).

UAI$_ENCRYPT2

When you specify UAI$_ENCRYPT2, $GETUAI returns a code indicating the encryption algorithm for the secondary password.

Because the encryption algorithm is a byte in length, the buffer length field in the item descriptor should specify 1 (byte).

UAI$_ENQLM

When you specify UAI$_ENQLM, $GETUAI returns the lock queue limit.

Because this decimal number is a word in length, the buffer length field in the item descriptor should specify 2 (bytes).

UAI$_EXPIRATION

When you specify UAI$_EXPIRATION, $GETUAI returns, as a quadword absolute time value, the expiration date and time of the account.

Because the absolute time value is a quadword in length, the buffer length field in the item descriptor should specify 8 (bytes).

UAI$_FILLM

When you specify UAI$_FILLM, $GETUAI returns the open file limit.

Because this decimal number is a word in length, the buffer length field in the item descriptor should specify 2 (bytes).

UAI$_FLAGS

When you specify UAI$_FLAGS, $GETUAI returns, as a longword bit vector, the various login flags set for the user.

Each flag is represented by a bit. The $UAIDEF macro defines the following symbolic names for these flags.

Symbolic Name	Description
UAI$V_AUDIT	All actions are audited.
UAI$V_AUTOLOGIN	User can only log in to terminals defined by the automatic login facility (ALF).
UAI$V_CAPTIVE	User is restricted to captive account.
UAI$V_DEFCLI	User is restricted to default command interpreter.
UAI$V_DISACNT	User account is disabled.
UAI$V_DISCTLY	User cannot use CTRL/Y.
UAI$V_DISMAIL	Announcement of new mail is suppressed.
UAI$V_DISRECONNECT	User cannot reconnect to existing processes.
UAI$V_DISREPORT	User will not receive last login mesages.
UAI$V_DISWELCOME	User will not receive the login welcome message.
UAI$V_FORCE_EXP_PWD_CHANGE	User is required to change expired passwords.
UAI$V_GENPWD	User is required to use generated passwords.
UAI$V_LOCKPWD	SET PASSWORD command is disabled.
UAI$V_NOMAIL	Mail delivery to user is disabled.
UAI$V_PWD_EXPIRED	Primary password is expired.
UAI$V_PWD2_EXPIRED	Secondary password is expired.

UAI$_JTQUOTA

When you specify UAI$_JTQUOTA, $GETUAI returns the initial byte quota with which the jobwide logical name table is to be created.

Because this quota is a longword decimal number, the buffer length field in the item descriptor should specify 4 (bytes).

UAI$_LASTLOGIN_I

When you specify UAI$_LASTLOGIN_I, $GETUAI returns, as a quadword absolute time value, the date of the last interactive login.

UAI$_LASTLOGIN_N

When you specify UAI$_LASTLOGIN_N, $GETUAI returns, as a quadword absolute time value, the date of the last noninteractive login.

UAI$_LGICMD

When you specify UAI$_LGICMD, $GETUAI returns, as an RMS file specification, the name of the default login command file.

Because a file specification can include up to 63 characters plus a size-byte prefix, the buffer length field of the item descriptor should specify 64 (bytes).

UAI$_LOCAL_ACCESS_P

When UAI$_LOCAL_ACCESS_P, $GETUAI returns, as a 3-byte value, the range of times during which local interactive access is permitted for primary days. Each bit set represents a 1-hour period, from bit 0 as midnight to 1 a.m. to bit 23 as 11 p.m. to midnight.

The buffer length field in the item descriptor should specify 3 (bytes).

UAI$_LOCAL_ACCESS_S

When you specify UAI$_LOCAL_ACCESS_S, $GETUAI returns, as a 3-byte value, the range of times during which batch access is permitted for secondary days. Each bit set represents a 1-hour period, from bit 0 as midnight to 1 a.m. to bit 23 as 11 p.m. to midnight.

The buffer length field in the item descriptor should specify 3 (bytes).

UAI$_LOGFAILS

When you specify UAI$_LOGFAILS, $GETUAI returns the count of login failures.

Because this decimal number is a word in length, the buffer length field in the item descriptor should specify 2 (bytes).

UAI$_MAXACCTJOBS

When you specify UAI$_MAXACCTJOBS, $GETUAI returns the maximum number of batch, interactive, and detached processes that may be active at one time for all users of the same account. The value 0 represents an unlimited number.

Because this decimal number is a word in length, the buffer length field in the item descriptor should specify 2 (bytes).

UAI$_MAXDETACH

When you specify UAI$_MAXDETACH, $GETUAI returns the detached process limit. A value of 0 represents an unlimited number.

Because this decimal number is a word in length, the buffer length field in the item descriptor should specify 2 (bytes).

UAI$_MAXJOBS

When you specify UAI$_MAXJOBS, $GETUAI returns the active process limit. A value of 0 represents an unlimited number.

Because this decimal number is a word in length, the buffer length field in the item descriptor should specify 2 (bytes).

UAI$_NETWORK_ACCESS_P

When you specify UAI$_NETWORK_ACCESS_P, $GETUAI returns, as a 3-byte value, the range of times during which network access is permitted for primary days. Each bit set represents a 1-hour period, from bit 0 as midnight to 1 a.m. to bit 23 as 11 p.m. to midnight.

The buffer length field in the item descriptor should specify 3 (bytes).

UAI$_NETWORK_ACCESS_S

When you specify UAI$_NETWORK_ACCESS_S, $GETUAI returns, as a 3-byte value, the range of times during which network access is permitted for secondary days. Each bit set represents a 1-hour period, from bit 0 as midnight to 1 a.m. to bit 23 as 11 p.m. to midnight.

The buffer length field in the item descriptor should specify 3 (bytes).

UAI$_OWNER

When you specify UAI$_OWNER, $GETUAI returns, as a character string, the name of the owner of the account.

Because the owner name can include up to 31 characters plus a size-byte prefix, the buffer length field of the item descriptor should specify 32 (bytes).

UAI$_PBYTLM

When you specify UAI$_PBYTLM, $GETUAI returns the paged buffer I/O byte count limit.

Because the paged buffer I/O byte count limit is a longword decimal number, the buffer length field in the item descriptor should specify 4 (bytes).

UAI$_PGFLQUOTA

When you specify UAI$_PGFLQUOTA, $GETUAI returns the paging file quota.

Because the paging file quota is a longword decimal number, the buffer length field in the item descriptor should specify 4 (bytes).

UAI$_PRCCNT

When you specify UAI$_PRCCNT, $GETUAI returns the subprocess creation limit.

Because the subprocess creation limit is a longword decimal number, the buffer length field in the item descriptor should specify 4 (bytes).

UAI$_PRI

When you specify UAI$_PRI, $GETUAI returns the default base priority in the range 0 through 31.

Because this decimal number is a byte in length, the buffer length field in the item descriptor should specify 1 (byte).

UAI$_PRIMEDAYS

When you specify UAI$_PRIMEDAYS, $GETUAI returns, as a longword bit vector, the primary and secondary days of the week.

Each bit represents a day of the week, with the bit clear representing a primary day and the bit set representing a secondary day. The $UAIDEF macro defines the following symbolic names for these bits:

UAI$V_MONDAY
UAI$V_TUESDAY
UAI$V_WEDNESDAY
UAI$V_THURSDAY
UAI$V_FRIDAY
UAI$V_SATURDAY
UAI$V_SUNDAY

UAI$_PRIV

When you specify UAI$_PRIV, $GETUAI returns, as a quadword value, the names of the privileges the user holds.

Because this value is a quadword in length, the buffer length field in the item descriptor should specify 8 (bytes).

UAI$_PWD

When you specify UAI$_PWD, $GETUAI returns, as a quadword value, the hashed primary password of the user.

Because this value is a quadword in length, the buffer length field in the item descriptor should specify 8 (bytes).

UAI$_PWD_DATE

When you specify UAI$_PWD_DATE, $GETUAI returns, as a quadword absolute time value, the date of the last password change.

Because this value is a quadword in length, the buffer length field in the item descriptor should specify 8 (bytes).

UAI$_PWD_LENGTH

When you specify UAI$_PWD_LENGTH, $GETUAI returns the minimum password length.

Because this decimal number is a byte in length, the buffer length field in the item descriptor should specify 1 (byte).

UAI$_PWD_LIFETIME

When you specify UAI$_PWD_LIFETIME, $GETUAI returns, as a quadword absolute time value, the password lifetime.

Because this value is a quadword in length, the buffer length field in the item descriptor should specify 8 (bytes).

UAI$_PWD2

When you specify UAI$_PWD2, $GETUAI returns, as a quadword value, the hashed secondary password of the user.

Because this value is a quadword in length, the buffer length field in the item descriptor should specify 8 (bytes).

UAI$_PWD2_DATE

When you specify UAI$_PWD2_DATE, $GETUAI returns, as a quadword absolute time value, the last date on which the secondary password was changed.

Because this value is a quadword in length, the buffer length field in the item descriptor should specify 8 (bytes).

UAI$_QUEPRI

When you specify UAI$_QUEPRI, $GETUAI returns the maximum job queue priority.

Because this decimal number is a byte in length, the buffer length field in the item descriptor should specify 1 (byte).

UAI$_REMOTE_ACCESS_P

When you specify UAI$_REMOTE_ACCESS_P, $GETUAI returns, as a 3-byte value, the range of times during which remote interactive access is permitted for primary days. Each bit set represents a 1-hour period, from bit 0 as midnight to 1 a.m. to bit 23 as 11 p.m. to midnight.

The buffer length field in the item descriptor should specify 3 (bytes).

UAI$_REMOTE_ACCESS_S

When you specify UAI$_REMOTE_ACCESS_S, $GETUAI returns, as a 3-byte value, the range of times during which remote interactive access is permitted for secondary days. Each bit set represents a 1-hour period, from bit 0 as midnight to 1 a.m. to bit 23 as 11 p.m. to midnight.

The buffer length field in the item descriptor should specify 3 (bytes).

UAI$_SALT

When you specify UAI$_SALT, $GETUAI returns the random password salt.

Because this decimal number is a word in length, the buffer length field in the item descriptor should specify 2 (bytes).

UAI$_SHRFILLM

When you specify UAI$_SHRFILLM, $GETUAI returns the shared file limit.

Because this decimal number is a word in length, the buffer length field in the item descriptor should specify 2 (bytes).

UAI$_TQCNT

When you specify UAI$_TQCNT, $GETUAI returns the timer queue entry limit.

Because this decimal number is a word in length, the buffer length field in the item descriptor should specify 2 (bytes).

UAI$_UIC

When you specify UAI$_UIC, $GETUAI returns, as a longword, the user identification code (UIC), containing the following two word-length subfields:

Symbolic Name	Description
UIC$W_MEM	The member number subfield of the UIC
UIC$W_GRP	The group number subfield of the UIC

UAI$_USERNAME

When you specify UAI$_USERNAME, $GETUAI returns the user name of the owner of the specified job.

Because a user name can include up to 12 characters, the buffer length field of the item descriptor should specify 12 (bytes).

UAI$_WSEXTENT

When you specify UAI$_WSEXTENT, $GETUAI returns the working set extent for the user of the specified queue or job.

Because the working set extent is a longword decimal number, the buffer length field in the item descriptor should specify 4 (bytes).

UAI$_WSQUOTA

When you specify UAI$_WSQUOTA, $GETUAI returns the working set quota for the specified user.

Because this quota is a longword decimal number, the buffer length field in the item descriptor should specify 4 (bytes).

DESCRIPTION

Use the following list to determine the privileges required to use the $GETUAI service:

- BYPASS or SYSPRV—Allows access to any record in the user authorization file (UAF)
- GRPPRV—Allows access to any record in the UAF whose UIC group matches that of the requester
- No privilege—Allows access to any UAF record whose UIC matches that of the requester

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The item list or input buffer cannot be read by the caller; or the return length buffer, output buffer, or status block cannot be written by the caller.
SS$_BADPARAM	The function code is invalid; the item list contains an invalid item code; a buffer descriptor has an invalid length; or the reserved parameter has a nonzero value.
SS$_NOPRIV	The user does not have the privileges required to examine the authorization information for the specified user.

This service may also return RMS status codes associated with operations on indexed files. For a description of RMS status codes that are returned by this service, refer to the *VMS Record Management Services Manual.*

$GRANTID Grant Identifier to Process

The Grant Identifier to Process service adds the specified identifier record to the rights list of the process or the system. If the identifier is already in the rights list, the attributes are modified as specified.

FORMAT

SYS$GRANTID *[pidadr] ,[prcnam] ,[id] ,[name] ,[prvatr]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

pidadr

VMS usage: **process_id**
type: **longword (unsigned)**
access: **modify**
mechanism: **by reference**

Process identification (PID) number of the process affected when $GRANTID completes execution. The **pidadr** argument is the address of longword containing the PID of the process to be affected. You use –1 to indicate the system rights list. When **pidadr** is passed, it is also returned; therefore, you must pass it as a variable rather than a constant. If you specify neither **pidadr** nor **prcnam**, your own process is used.

prcnam

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

Process name on which $GRANTID operates. The **prcnam** argument is the address of a character string descriptor containing the process name. The maximum length of the name is 15 characters. Because the UIC group number is interpreted as part of the process name, you must use **pidadr** to specify the rights list of a process in a different group. If you specify neither **pidadr** nor **prcnam**, your own process is used.

id

VMS usage: **rights_holder**
type: **quadword (unsigned)**
access: **modify**
mechanism: **by reference**

Identifier and attributes to be granted when $GRANTID completes execution. The **id** argument is the address of a quadword containing the binary identifier

code to be granted in the first longword and the attributes in the second longword.

Symbol values are offsets to the bits within the longword. You can also obtain the values as masks with the appropriate bit set using the prefix KGB$M rather than KGB$V. The following symbols for each bit position are defined in the macro library ($KGBDEF).

Bit Position	Meaning When Set
KGB$V_DYNAMIC	Allows the unprivileged holder to add or remove the identifier from the process rights list
KGB$V_RESOURCE	Allows the holder to charge resources, such as disk blocks, to the identifier

You must specify either **id** or **name**. Because the **id** argument is returned as well as passed if you specify **name**, you must pass it as a variable rather than a constant in this case.

name

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

Name of the identifier granted when $GRANTID completes execution. The **name** argument is the address of a descriptor pointing to the name of the identifier. You must specify either **id** or **name**.

prvatr

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

Previous attributes of the identifier. The **prvatr** argument is the address of a longword used to store the attributes of the identifier if it was previously present in the rights list. If you added rather than modified the identifier, **prvatr** is ignored.

DESCRIPTION

The Grant Identifier to Process service adds the specified identifier to the rights list of the process or the system. If the identifier is already in the rights list, its attributes are modified to those specified. This service is meant to be used by a privileged subsystem to alter the access rights profile of a user, based on installation policy. It is not meant to be used by the general system user.

You need CMKRNL privilege to invoke this service. In addition, you need GROUP privilege to modify the rights list of a process in the same group as the calling process (unless the process has the same UIC as the calling process). You need WORLD privilege to modify the rights list of a process outside the caller's group. You need SYSNAM privilege to modify the system rights list.

The result of passing the **pidadr** or the **prcnam** argument, or both, to SYS$GRANTID is summarized in the following table:

prcnam	pidadr	Result
Omitted	Omitted	Current process ID is used; process ID is not returned.
Omitted	0	Current process ID is used; process ID is returned.
Omitted	Specified	Specified process ID is used; process ID is returned.
Specified	Omitted	Specified process name is used; process ID is not returned.
Specified	0	Specified process name is used; process ID is returned.
Specified	Specified	Specified process ID is used, process ID is returned, and process name is ignored.

The result of passing the **name** or the **id** argument, or both, to SYS$GRANTID is summarized in the following table:

name	id	Result
Omitted	Omitted	Illegal.
Omitted	Specified	Specified identifier value is used; identifier value is returned.
Specified	Omitted	Specified identifier name is used; identifier value is not returned.
Specified	0	Specified identifier name is used; identifier value is returned.
Specified	Specified	Specified identifier value is used, identifier value is returned, and identifier name is ignored.

CONDITION VALUES RETURNED

SS$_WASCLR	The service completed successfully; the rights list did not contain the specified identifier.
SS$_WASSET	The service completed successfully; the rights list already held the specified identifier.
SS$_ACCVIO	The **pidadr** argument cannot be read or written, or **prcnam** cannot be read, or **id** cannot be read or written, or the name cannot be read, or **prvatr** cannot be written.
SS$_IVIDENT	The specified identifier or holder is of invalid format, or the specified identifier and holder are equal.
SS$_INSFMEM	The process dynamic memory is insufficient for opening the rights database.

SS$_NOPRIV	The caller does not have CMKRNL privilege, or is not running in executive or kernel mode, or the caller lacks GROUP, WORLD, or SYSNAM privilege as required.
SS$_NOSUCHID	The specified identifier name does not exist in the rights database. Note that the binary identifier, if given, is not validated against the rights database.
SS$_RIGHTSFULL	The rights list of the process or system is full.
SS$_NOSYSNAM	The operation requires SYSNAM privilege.
SS$_IVLOGNAM	You specified an invalid logical name.
SS$_NONEXPR	You specified a nonexistent process.
RMS$_PRV	The user does not have read access to the rights database.

Because the rights database is an indexed file accessed with VMS RMS, this service may also return RMS status codes associated with operations on indexed files. For descriptions of these status codes, refer to the *VMS Record Management Services Manual.*

$HIBER Hibernate

The Hibernate service allows a process to make itself inactive but to remain known to the system so that it can be interrupted, for example, to receive ASTs. A hibernate request is a wait-for-wake-event request. When you call the Wake Process from Hibernation ($WAKE) service or when the time specified with the Schedule Wakeup ($SCHDWK) service occurs, the process continues execution at the instruction following the Hibernate call.

FORMAT

SYS$HIBER

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

None.

DESCRIPTION

In VAX MACRO, you can call the Hibernate service only by using the $name_S macro.

A hibernating process can be swapped out of the balance set if it is not locked into the balance set.

An AST can interrupt the wait state caused by $HIBER if the access mode at which the AST is to execute is equal to or more privileged than the access mode from which the hibernate request was issued and the process is enabled for ASTs at that access mode.

When the AST service routine completes execution, the system reexecutes the $HIBER service on behalf of the process. If a wakeup request has been issued for the process during the execution of the AST service routine (either by itself or another process), the process resumes execution. If a wakeup request has not been issued, it continues to hibernate.

If one or more wakeup requests are issued for the process while it is not hibernating, the next hibernate call returns immediately; that is, the process does not hibernate. No count is maintained of outstanding wakeup requests.

Although this service has no arguments, a FORTRAN function reference must use parentheses to indicate a null argument list, as in the following example:

```
ISTAT=SYS$HIBER()
```

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.

$IDTOASC Translate Identifier to Identifier Name

The Translate Identifier to Identifier Name service translates the specified identifier value to its identifier name.

FORMAT

SYS$IDTOASC *id ,[namlen] ,[nambuf] ,[resid] ,[attrib] ,[contxt]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

id

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

Binary identifier value translated by $IDTOASC. The **id** argument is a longword containing the binary value of the identifier. To determine the identifier names of all identifiers in the rights database, you specify **id** as –1 and call SYS$IDTOASC repeatedly until it returns the status code SS$_NOSUCHID. The identifiers are returned in alphabetical order.

namlen

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

Number of characters in the identifier name translated by $IDTOASC. The **namlen** argument is the address of a word containing the length of the identifier name written to **nambuf**.

nambuf

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

Identifier name text string returned when $IDTOASC completes the translation. The **nambuf** argument is the address of a descriptor pointing to the buffer in which the identifier name is written.

resid

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

Identifier value of the identifier name returned in **nambuf**. The **resid** argument is the address of a longword containing the 32-bit code of the identifier.

attrib

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

Mask of attributes associated with the identifier returned in **resid**. The **attrib** argument is the address of a longword containing the attribute mask.

Symbol values are offsets to the bits within the longword. You can also obtain the values as masks with the appropriate bit set using the prefix KGB$M rather than KGB$V. The following symbols for each bit position are defined in the system macro library ($KGBDEF).

Bit Position	Meaning When Set
KGB$V_DYNAMIC	Allows the unprivileged holder to add or remove the identifier from the process rights list
KGB$V_RESOURCE	Allows the holder to charge resources, such as disk blocks, to the identifier

contxt

VMS usage: **context**
type: **longword (unsigned)**
access: **modify**
mechanism: **by reference**

Context value used when repeatedly calling $IDTOASC. The **contxt** argument is the address of a longword used while $IDTOASC searches for all identifiers. The context value must be initialized to zero, and the resulting context of each call to $IDTOASC must be presented to each subsequent call. After **contxt** is passed to $IDTOASC, you must not modify its value.

DESCRIPTION

The Translate Identifier to Identifier Name service translates the specified binary identifier value to an identifier name. While the primary purpose of this service is to translate the specified identifier to its name, you may also use it to find all identifiers in the rights database. To determine all the identifiers, call $IDTOASC repeatedly until it returns the status code SS$_NOSUCHID. When SS$_NOSUCHID is returned, $IDTOASC has returned all the identifiers, cleared the context value, and deallocated the record stream.

If you complete your calls to $IDTOASC before SS$_NOSUCHID is returned, use SYS$FINISH_RDB to clear the context value and deallocate the record stream.

When you use wildcards with this service, the records are returned in identifier name order.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The **namlen**, **nambuf**, **resid**, **attrib**, or **contxt** argument cannot be written by the caller.
SS$_INSFMEM	The process dynamic memory is insufficient for opening the rights database.
SS$_IVCHAN	The contents of the context longword are not valid.
SS$_IVIDENT	The specified identifier is of invalid format.
SS$_NOIOCHAN	No more rights database context streams are available.
SS$_NOSUCHID	The specified identifier name does not exist in the rights database, or the entire rights database has been searched if the ID is −1.
RMS$_PRV	The user does not have read access to the rights database.

Because the rights database is an indexed file that you access with VMS RMS, this service may also return RMS status codes associated with operations on indexed files. For descriptions of these status codes, refer to the *VMS Record Management Services Manual.*

$LCKPAG Lock Pages in Memory

The Lock Pages In Memory service locks a page or range of pages in memory. The specified virtual pages are forced into the working set and then locked in memory. A locked page is not swapped out of memory if the working set of the process is swapped out. These pages are not candidates for page replacement and in this sense are locked in the working set as well.

FORMAT

SYS$LCKPAG *inadr ,[retadr] ,[acmode]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

inadr

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

Starting and ending virtual addresses of the range of pages to be locked. The **inadr** argument is the address of a 2-longword array containing, in order, the starting and ending process virtual addresses. Only the virtual page number portion of each virtual address is used; the low-order 9 bits are ignored.

If the starting and ending virtual addresses are the same, a single page is locked.

retadr

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

Starting and ending process virtual addresses of the pages that $LCKPAG actually locked. The **retadr** argument is the address of a 2-longword array containing, in order, the starting and ending process virtual addresses.

acmode

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

Access mode to be associated with the pages to be locked. The **acmode** argument is a longword containing the access mode. The $PSLDEF macro defines the four access modes.

The most privileged access mode used is the access mode of the caller. For the $LCKPAG service to complete successfully, the resultant access mode must be equal to or more privileged than the access mode already associated with the pages to be locked.

DESCRIPTION

The calling process must have PSWAPM privilege to lock pages into memory.

If more than one page is being locked and you need to determine specifically which pages were previously locked, the pages should be locked one at a time.

If an error occurs while the $LCKPAG service is locking pages, the return array, if requested, indicates the pages that were successfully locked before the error occurred. If no pages are locked, both longwords in the return address array contain the value -1.

You can unlock pages locked in memory with the Unlock Pages from Memory ($ULKPAG) service. Locked pages are automatically unlocked at image exit.

CONDITION VALUES RETURNED

SS$_WASCLR	The service completed successfully. All of the specified pages were previously unlocked.
SS$_WASSET	The service completed successfully. At least one of the specified pages was previously locked.
SS$_ACCVIO	The input array cannot be read by the caller; the output array cannot be written by the caller; or a page in the specified range is inaccessible or does not exist.
SS$_LCKPAGFUL	The system-defined maximum limit on the number of pages that can be locked in memory has been reached.
SS$_NOPRIV	The process does not have the privilege to lock pages in memory.

$LKWSET Lock Pages in Working Set

The Lock Pages in Working Set service locks a range of pages in the working set; if the pages are not already in the working set, it brings them in and locks them. A page locked in the working set does not become a candidate for replacement.

FORMAT

SYS$LKWSET *inadr ,[retadr] ,[acmode]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

inadr

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

Starting and ending virtual addresses of the range of pages to be locked in the working set. The **inadr** argument is the address of a 2-longword array containing, in order, the starting and ending process virtual addresses. Only the virtual page number portion of each virtual address is used; the low-order 9 bits are ignored.

If the starting and ending virtual addresses are the same, a single page is locked.

retadr

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

Starting and ending process virtual addresses of the range of pages actually locked by $LCKWSET. The **retadr** argument is the address of a 2-longword array containing, in order, the starting and ending process virtual addresses.

acmode

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

Access mode to be associated with the pages to be locked. The **acmode** argument is a longword containing the access mode. The $PSLDEF macro defines the four access modes.

The most privileged access mode used is the access mode of the caller. For the $LKWSET service to complete successfully, the resultant access mode must be equal to or more privileged than the access mode already associated with the pages to be locked.

DESCRIPTION

If more than one page is being locked and you need to determine specifically which pages were previously locked, the pages should be locked one at a time.

If an error occurs while the $LKWSET service is locking pages, the return array, if requested, indicates the pages that were successfully locked before the error occurred. If no pages are locked, both longwords in the return address array contain -1.

You can unlock pages locked in the working set with the Unlock Page from Working Set ($ULWSET) service.

Global pages with write access cannot be locked into the working set.

CONDITION VALUES RETURNED

SS$_WASCLR	The service completed successfully. All of the specified pages were previously unlocked.
SS$_WASSET	The service completed successfully. At least one of the specified pages was previously locked in the working set.
SS$_ACCVIO	The input address array cannot be read by the caller; the output address array cannot be written by the caller; or a page in the specified range is inaccessible or nonexistent.
SS$_LKWSETFUL	The locked working set is full. If any more pages are locked, not enough dynamic pages will be available to continue execution.
SS$_NOPRIV	A page in the specified range is in the system address space, or a global page with write access was specified.
SS$_PAGOWNVIO	The pages could not be locked because the access mode associated with the call to $LKWSET was less privileged than the access mode associated with the pages that were to be locked.

$MGBLSC Map Global Section

The Map Global Section service establishes a correspondence between pages (maps) in the virtual address space of the process and physical pages occupied by a global section.

FORMAT

SYS$MGBLSC *inadr ,[retadr] ,[acmode] ,[flags] ,gsdnam ,[ident] ,[relpag]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

inadr

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

Starting and ending virtual addresses in the virtual address space of the process (either the P0 or P1 regions) into which the section is to be mapped. The **inadr** argument is the address of a 2-longword array containing, in order, the starting and the ending process virtual addresses. Only the virtual page number portion of each virtual address is used; the low-order 9 bits are ignored.

If the starting and ending virtual addresses are the same, a single page is mapped (except when the SEC$M_EXPREG bit is set in the **flags** argument).

If the SEC$M_EXPREG bit is set in the **flags** argument, the starting address (first longword) specified in the **inadr** argument determines only whether the section is mapped in the program (P0) region or control (P1) region; the ending address (second longword) is ignored.

retadr

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

Starting and ending process virtual addresses into which the section was actually mapped by $MGBLSC. The **retadr** argument is the address of a 2-longword array containing, in order, the starting and ending process virtual addresses.

acmode

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

Access mode to be associated with the pages mapped into the process virtual address space. The **acmode** argument is a longword containing the access mode. The $PSLDEF macro defines symbols for the four access modes.

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

flags

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

Flag mask specifying options for the operation. The **flags** argument is a longword bit vector wherein a bit, when set, specifies the corresponding option.

The $SECDEF macro defines symbolic names for the flag bits. You construct the **flags** argument by specifying the symbolic names of each desired option in a logical OR operation. The following table describes each flag option.

Flag	Description
SEC$M_WRT	Map section with read/write access. By default, the section is mapped with read-only access.
SEC$M_SYSGBL	Map a system global section. By default, the section is a group global section.
SEC$M_EXPREG	Map the section in the first available virtual address range. By default, the section is mapped into the range specified by the **inadr** argument.

gsdnam

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

Name of the global section. The **gsdnam** argument is the address of a character string descriptor pointing to this name string.

For group global sections, VMS interprets the group UIC as part of the global section name; thus, the names of global sections are unique to UIC groups. Further, all global section names are implicitly qualified by their identification fields.

ident

VMS usage: **section_id**
type: **quadword (unsigned)**
access: **read only**
mechanism: **by reference**

Identification value specifying the version number of a global section, and, for processes mapping to an existing global section, the criteria for matching the identification. The **ident** argument is the address of a quadword structure containing three fields.

The first longword specifies, in the low-order 3 bits, the matching criteria. Their valid values, the symbolic names by which they can be specified, and their meanings are as follows.

Value/Name	Match Criteria
0 SEC$K_MATALL	Match all versions of the section.
1 SEC$K_MATEQU	Match only if major and minor identifications match.
2 SEC$K_MATLEQ	Match if the major identifications are equal and the minor identification of the mapper is less than or equal to the minor identification of the global section.

The version number is in the second longword and contains two fields: a minor identification in the low-order 24 bits and a major identification in the high-order 8 bits.

If you do not specify **ident** or specify it as 0 (the default), the version number and match control fields default to 0.

relpag

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

Relative page number within the section of the first page to be mapped. The **relpag** argument is a longword containing this number.

If you do not specify **relpag** or specify it as 0 (the default), the global section is mapped beginning with the first virtual block in the section.

DESCRIPTION

The protection mask specified at the time the global section is created determines the type of access (for example, read/write or read/only) that a particular process has to the section.

The $MGBLS service uses the following system resources:

- The working set limit quota (WSQUOTA) of the process must be sufficient to accommodate the increased size of the virtual address space when the $MGBLSC service maps a section.
- If the section pages are copy-on-reference, the process must also have sufficient paging file quota (PGFLQUOTA).

$MGBLSC

This system service causes the working set of the calling process to be adjusted to the size specified by the working set quota (WSQUOTA). If the working set size of the process is less than quota, the working set size is increased; if the working set size of the process is greater than quota, the working set size is decreased.

When $MGBLSC maps a global section, it adds pages to the virtual address space of the process. The section is mapped from a low address to a high address, whether the section is mapped in the program or control region.

If an error occurs during the mapping of a global section, the return address array, if specified, indicates the pages that were successfully mapped when the error occurred. If no pages were mapped, both longwords of the return address array contain −1.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The input address array, the global section name or name descriptor, or the section identification field cannot be read by the caller; or the return address array cannot be written by the caller.
SS$_ENDOFFILE	The starting virtual block number specified is beyond the logical end-of-file.
SS$_EXQUOTA	The process exceeded its paging file quota, creating copy-on-reference pages.
SS$_INSFWSL	The working set limit of the process is not large enough to accommodate the increased virtual address space.
SS$_INTERLOCK	The bit map lock for allocating global sections from the specified shared memory is locked by another process.
SS$_IVLOGNAM	The global section name has a length of 0 or has more than 15 characters.
SS$_IVSECFLG	You set a reserved flag.
SS$_IVSECIDCTL	The match control field of the global section identification is invalid.
SS$_NOPRIV	The file protection mask specified when the global section was created prohibits the type of access requested by the caller; or a page in the input address range is in the system address space.
SS$_NOSUCHSEC	The specified global section does not exist.
SS$_PAGOWNVIO	A page in the specified input address range is owned by a more privileged access mode.
SS$_SHMNOTCNCT	The shared memory named in the **gsdnam** argument is not known to the system. This error can be caused by a spelling error in the string, an improperly assigned logical name, or the failure to identify the memory as shared at system generation time.

SS$_TOOMANYLNAM	Logical name translation of the **gsdnam** string exceeded the allowed depth.
SS$_VASFULL	The virtual address space of the process is full; no space is available in the page tables for the pages created to contain the mapped global section.

$MOD_HOLDER Modify Holder Record in Rights Database

The Modify Holder Record in Rights Database service modifies the specified holder record of the target identifier in the rights database. Identifier attributes may be added or removed, or both.

FORMAT

SYS$MOD_HOLDER *id ,holder ,[set_attrib] ,[clr_attrib]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

id

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

Binary value of target identifier whose holder record is modified when $MOD_HOLDER completes execution. The **id** argument is a longword containing the identifier value.

holder

VMS usage: **rights_holder**
type: **quadword (unsigned)**
access: **read only**
mechanism: **by reference**

Identifier of holder being modified when $MOD_HOLDER completes execution. The **holder** argument is the address of a quadword containing the UIC identifier of the holder in the first longword and the value of zero in the second longword.

set_attrib

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

Bit mask of attributes to be enabled for the identifier when $MOD_HOLDER completes execution. The **set_attrib** argument is a longword containing the attribute mask.

The attributes actually enabled are the intersection of those specified and the attributes of the identifier. If you specify the same attribute in **set_attrib** and **clr_attrib**, the attribute is enabled.

Symbol values are offsets to the bits within the longword. You can also obtain the values as masks with the appropriate bit set using the prefix KGB$M rather than KGB$V. The following symbols for each bit position are defined in the system macro library ($KGBDEF).

Bit Position	Meaning When Set
KGB$V_DYNAMIC	Allows the unprivileged holder to add or remove the identifier from the process rights list.
KGB$V_RESOURCE	Allows the holder to charge resources, such as disk blocks, to the identifier.

clr_attrib

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

Bit mask of attributes to be disabled for the identifier when $MOD_HOLDER completes execution. The **clr_attrib** argument is a longword containing the attribute mask.

If you specify the same attribute in **set_attrib** and **clr_attrib**, the attribute is enabled.

Symbol values are offsets to the bits within the longword. You can also obtain the values as masks with the appropriate bit set using the prefix KGB$M rather than KGB$V. The following symbols for each bit position are defined in the system macro library ($KGBDEF).

Bit Position	Meaning When Set
KGB$V_DYNAMIC	Allows the unprivileged holder to add or remove the identifier from the process rights list.
KGB$V_RESOURCE	Allows the holder to charge resources, such as disk blocks, to the identifier.

DESCRIPTION

The Modify Holder Record In Rights Database service modifies the specified holder record in the rights database. Identifier attributes may be added or removed, or both.

When you specify both the **set_attrib** and **clr_attrib** arguments, the attribute is cleared first. Thus, if you specify the same attribute bit with each argument, the result is that the bit is set.

You need write access to the rights database to use this service. If the database is in SYS$SYSTEM (the default), you need SYSPRV privilege to grant write access to the database.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The **holder** argument cannot be read by the caller.
SS$_BADPARAM	The specified attributes contain invalid attribute flags.
SS$_INSFMEM	The process dynamic memory is insufficient for opening the rights database.
SS$_IVIDENT	The specified identifier or holder identifier is of invalid format.
SS$_NOSUCHID	The specified identifier does not exist in the rights database, or the specified holder identifier does not exist in the rights database.
RMS$_PRV	The user does not have write access to the rights database.

Because the rights database is an indexed file accessed with VMS RMS, this service may also return RMS status codes associated with operations on indexed files. For descriptions of these status codes, refer to the *VMS Record Management Services Manual.*

$MOD_IDENT Modify Identifier in Rights Database

The Modify Identifier in Rights Database service modifies the specified identifier record in the rights database. Identifier attributes may be added or removed, or both. The identifier name or value may be changed.

FORMAT

SYS$MOD_IDENT *id ,[set_attrib] ,[clr_attrib] ,[new_name] ,[new_value]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

id

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

Binary value of identifier whose identifier record is modified when $MOD_IDENT completes execution. The **id** argument is a longword containing the identifier value.

set_attrib

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

Bit mask of attributes to be enabled for the identifier when $MOD_IDENT completes execution. The **set_attrib** argument is a longword containing the attribute mask.

The attributes actually enabled are the intersection of those specified and the attributes of the identifier. If you specify the same attribute in **set_attrib** and **clr_attrib**, the attribute is enabled.

Symbol values are offsets to the bits within the longword. You can also obtain the values as masks with the appropriate bit set using the prefix KGB$M rather than KGB$V. The following symbols for each bit position are defined in the system macro library ($KGBDEF).

Bit Position	Meaning When Set
KGB$V_DYNAMIC	Allows the unprivileged holder to add or remove the identifier from the process rights list.
KGB$V_RESOURCE	Allows the holder to charge resources, such as disk blocks, to the identifier.

clr_attrib

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

Bit mask of attributes to be disabled for the identifier when $MOD_IDENT completes execution. The **clr_attrib** argument is a longword containing the attribute mask.

If you specify the same attribute in **set_attrib** and **clr_attrib**, the attribute is enabled.

Symbol values are offsets to the bits within the longword. You can also obtain the values as masks with the appropriate bit set using the prefix KGB$M rather than KGB$V. The following symbols for each bit position are defined in the system macro library ($KGBDEF).

Bit Position	Meaning When Set
KGB$V_DYNAMIC	Allows the unprivileged holder to add or remove the identifier from the process rights list.
KGB$V_RESOURCE	Allows the holder to charge resources, such as disk blocks, to the identifier.

new_name

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

New name to be given to the specified identifier. The **new_name** argument is the address of the descriptor pointing to the identifier name string.

An identifier name consists of 1 to 31 alphanumeric characters including dollar signs ($) and underscores (_), and must contain at least 1 nonnumeric character. Any lowercase characters specified are automatically converted to uppercase.

new_value

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

New value to be assigned to the specified identifier. The **new_value** argument is a longword containing the binary value of the specified identifier. When the identifier value is changed, $MOD_IDENT also changes the value of the identifier in all of the holder records in which the specified identifier appears.

DESCRIPTION

The Modify Identifier in Rights Database service modifies the specified identifier record in the rights database. When you specify both the **set_attrib** and **clr_attrib** arguments, the attribute is cleared first. Thus, if you specify the same attribute bit with each argument, the result is that the bit is set.

You need write access to the rights database to use this service. If the database is in SYS$SYSTEM (the default) you need SYSPRV privilege to grant write access to the database.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_NOSUCHID	The specified identifier does not exist in the rights database.
SS$_BADPARAM	The specified attributes contain invalid attribute flags.
SS$_INSFMEM	The process dynamic memory is insufficient for opening the rights database.
SS$_IVIDENT	The specified identifier is of invalid format.
RMS$_PRV	The user does not have write access to the rights database.

Because the rights database is an indexed file accessed with VMS RMS, this service may also return RMS status codes associated with operations on indexed files. For descriptions of these status codes, refer to the *VMS Record Management Services Manual.*

$MOUNT Mount Volume

The Mount Volume service mounts a tape, disk volume, or volume set and specifies options for the mount operation.

FORMAT

SYS$MOUNT *itmlst*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENT

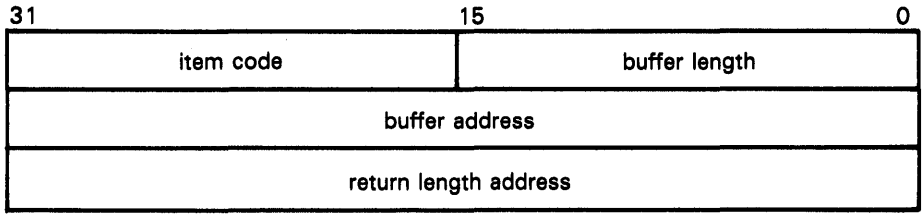
itmlst

VMS usage: **item_list_3**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Item list specifying options for the mount operation. The **itmlst** argument is the address of a list of item descriptors, each of which specifies an option and provides the information needed to perform the operation.

The item list must include at least one device item descriptor, and is terminated by a longword of 0.

The following diagram depicts the format of a single item descriptor.



ZK-1705-84

$MOUNT Item Descriptor Fields

buffer length
A word specifying the length (in bytes) of the buffer that supplies the information $MOUNT needs to process the specified item code. The required length of the buffer depends upon the item code specified in the **item code** field of the item descriptor. If the value of **buffer length** is too small, $MOUNT truncates the data.

item code

A word containing a user-supplied symbolic code that specifies an option for the mount operation. The $MNTDEF macro defines these codes, which are described in the $MOUNT Item Codes section.

buffer address

A longword containing the address of the buffer that supplies information to $MOUNT.

return length address

This field is not used.

$MOUNT Item Codes

MNT$_ACCESSED

The MNT$_ACCESSED item code specifies the number of directories that will be in use, concurrently, on the volume. The buffer must contain a longword integer value in the range 0 to 255. This value overrides the number of directories specified when the volume was initialized. To specify MNT$_ACCESSED, the caller must have OPER privilege. The MNT$_ACCESSED item code applies only to disks.

MNT$_BLOCKSIZE

The MNT$_BLOCKSIZE item code specifies the default block size for tape volumes. The buffer must contain a longword integer value in the range 20 to 65,532 bytes for VMS RMS operations or 10 to 65,534 bytes for operations that do not use VMS RMS. The MNT$_BLOCKSIZE item code applies only to tapes.

If you do not specify MNT$_BLOCKSIZE, the default block size is 2048 bytes for Files–11 tape volumes and 512 bytes for foreign and unlabeled tapes.

You must specify MNT$_BLOCKSIZE when mounting (1) tapes that do not have ANSI HDR2 labels, (2) tapes to which data will be written from compatibility mode, and (3) tapes that are to contain records whose size is larger than the default value.

MNT$_COMMENT

The MNT$_COMMENT item code specifies text to be associated with an operator request. The buffer must contain a character string of no more than 78 characters. This text will be printed on the operator's console if an operator request is issued for the device being mounted.

MNT$_DENSITY

The MNT$_DENSITY item code specifies the density at which data is to be written to a foreign or unlabeled tape. The buffer must contain a longword value that specifies one of the following legal densities: 800, 1600, or 6250 bpi. The MNT$_DENSITY item code applies only to tapes.

The specified density will be used only if (1) the tape is foreign or unlabeled and (2) the first operation is a write.

MNT$_DEVNAM

The MNT$_DEVNAM item code specifies the name of the device to be mounted. The buffer must contain a character string of from 1 to 64 characters, which is the device name. The device name may be a physical

device name or a logical name; if it is a logical name, it must translate to a physical device name.

The MNT$_DEVNAM item code must appear at least once in an item list, and it may appear more than once. It appears more than once when a volume set is being mounted, because, in this case, one device is being mounted for each volume in the volume set.

MNT$_EXTENSION

The MNT$_EXTENSION item code specifies the number of blocks by which files will be extended. The buffer must contain a longword value in the range 0 to 65,535. The MNT$_EXTENSION item code applies only to disks.

MNT$_EXTENT

The MNT$_EXTENT item code specifies the size of the extent cache in units of extent pointers. The buffer must contain a longword value, which specifies this size. To specify MNT$_EXTENT, you need OPER privilege. The value 0 (the default) disables caching. The MNT$_EXTENT item code applies only to disks.

MNT$_FILEID

The MNT$_FILEID item code specifies the size of the file-ID cache in units of file numbers. The buffer must contain a longword value, which specifies this size. To specify MNT$_FILEID, you need OPER privilege. The value 1 disables caching. The MNT$_FILEID item code applies only to disks.

MNT$_FLAGS

The MNT$_FLAGS item code specifies a longword bit vector wherein each bit specifies an option for the mount operation. The buffer must contain a longword, which is the bit vector.

The $MNTDEF macro defines symbolic names for each option (bit) in the bit vector. You construct the bit vector by specifying the symbolic names for the desired options in a logical OR operation. The following table describes the symbolic names for each option.

Option	Description
MNT$M_FOREIGN	The volume is to be mounted as a foreign volume; a foreign volume is not Files–11 structured. If you specify MNT$M_FOREIGN, the following item codes may each appear in the item list only once: MNT$_DEVNAM, MNT$_VOLNAM, and MNT$_LOGNAM. To specify MNT$M_FOREIGN, the caller must either own the volume or have VOLPRO privilege.
MNT$M_GROUP	The logical name for the volume to be mounted is entered in the group logical name table, and the volume is made accessible to other users with the same UIC group number as that of the calling process. To specify MNT$M_GROUP, the caller must have GRPNAM privilege. MNT$M_GROUP applies only to disks.
MNT$M_MULTI_VOL	Specifies, for foreign or unlabeled magnetic tapes, that subsequent volumes can be processed by overriding MOUNT's access checks. You can use this option when a utility that supports multivolume magnetic tape sets needs to process subsequent volumes, and these volumes do not contain labels that MOUNT can interpret. You need VOLPRO privilege to specify the MNT$M_MULTI_VOL option. This option can only be used together with the MNT$M_FOREIGN option.
MNT$M_NOASSIST	$MOUNT does not request operator assistance if errors are encountered during the mount operation. If not specified, $MOUNT requests operator assistance to recover from some error conditions.
MNT$M_NODISKQ	Disk quotas are not to be enforced for the volume to be mounted. If not specified, disk quotas are enforced. To specify MNT$M_NODISKQ, the caller must either own the volume or have VOLPRO privilege. MNT$M_NODISKQ applies only to disks.
MNT$M_NOHDR3	ANSI HDR3 and HDR4 labels are not to be written to magnetic tapes as they are mounted. If not specified, ANSI HDR3 and HDR4 labels are written to all tapes. Use MNT$M_NOHDR3 when writing to volumes that will be read by a system, such as the RT–11 system, which does not process HDR3 and HDR4 labels correctly. MNT$M_NOHDR3 applies only to tapes.

Option	Description
MNT$M_NOWRITE	The volume to be mounted is software write locked. If not specified, the volume is assumed to have read and write access.
MNT$M_OVR_ACCESS	If the installation allows, this option overrides any character in the Accessibility Field of the volume. The necessity of this option is defined by the installation. That is, each installation has the option of specifying a routine that the magnetic tape file system will use to process this field. By default, VMS provides a routine that checks this field in the folllowing manner: • If the magnetic tape was created on a version of VMS that conforms to Version 3 of ANSI, then you must use this option to override any character other than an ASCII space. • If a VMS protection is specified and that magnetic tape conforms to an ANSI standard that is higher than Version 3, then you must use this option to override any character other than an ASCII 1. To specify MNT$M_OVR_ACCESS, the caller must either own the volume or have VOLPRO privilege. MNT$M_OVR_ACCESS applies only to tapes.
MNT$M_OVR_EXP	A tape that has not yet reached its expiration date may be overwritten. To specify MNT$M_OVR_EXP, the caller must own the volume or have VOLPRO privilege.
MNT$M_OVR_IDENT	You can mount the volume without specifying the volume name (by using the MNT$_VOLNAM item code). If specified, the following options must not be specified: MNTM_GROUP, MNTM_SHARE, and MNT$M_SYSTEM.
MNT$M_OVR_LOCK	The software write lock that occurs when a volume has a corrupted storage bit mask may be overridden.
MNT$M_OVR_SETID	Checks on the volume set identification are not to be performed when subsequent reels in the volume set are mounted. MNT$M_OVR_SETID applies only to tapes.
MNT$M_READCHECK	Read checks are to be performed following all read operations.

Option	Description
MNT$M_SHARE	Volume is to be mounted shared and is therefore accessible to other users. MNT$M_SHARE applies only to disks. If the volume was previously mounted shared by another user and MNT$M_SHARE is specified in the current call, all other options specified in the current call are ignored. If the caller allocated the device and specified MNT$M_SHARE in the call to $MOUNT, $MOUNT will deallocate the device so that other users may access the volume.
MNT$M_MESSAGE	Messages will be sent to the caller's SYS$OUTPUT device.
MNT$M_SYSTEM	The logical name for the volume to be mounted is entered in the system logical name table, and the volume is made accessible to all other users, provided that UIC-based protection allows access to the volume. To specify MNT$M_SYSTEM, the caller must have SYSNAM privilege. MNT$M_SYSTEM applies only to disks.
MNT$M_WRITECHECK	Write checks are to be performed after all write operations.
MNT$M_WRITETHRU	Write-back caching is disabled so that file headers are written back to disk with every write operation. If not specified, file headers are cached until the file is closed. Caching file headers improves performance at the risk of losing written data if the system fails. MNT$M_WRITETHRU applies only to disks.
MNT$M_NOMNTVER	The volume is not marked as a candidate for automatic mount verification. If not specified, the volume is marked as a candidate for mount verification. MNT$M_NOMNTVER applies only to disks.
MNT$M_NOCACHE	All caching associated with the volume is turned off. Specifying MNT$M_NOCACHE is equivalent to (1) specifying MNT$M_WRITETHRU, (2) specifying a value of 1 for the item descriptor MNT$_FILEID, and (3) specifying a value of 0 for the item descriptors MNT$M_EXTENT and MNT$M_QUOTA. MNT$M_NOCACHE applies only to disks.

Option	Description
MNT$M_NOAUTO	Automatic volume labeling (AVL) and automatic volume recognition (AVR) are to be disabled. If MNT$M_NOAUTO is specified, the operator must enter commands from the console to process each additional volume in a volume set. When a volume is finished processing, the operator specifies the drive on which the next volume is loaded and the label name of the next volume. You may want to use MNT$M_NOAUTO to disable AVL and AVR when not reading a volume set sequentially. You can enable AVL and AVR by specifying MNT$M_INIT_CONT. MNT$M_NOAUTO applies only to magnetic tapes.
MNT$M_INIT_CONT	Additional volumes in the volume set are to be initialized without operator intervention. $MOUNT initializes new volumes with the protections specified for the first magnetic tape of the volume set and creates unique volume label names for up to 99 volumes in a volume set. If MNT$M_INIT_CONT is specified, you must allocate multiple magnetic tape drives to the volume set. If $MOUNT switches to a drive that has no magnetic tape loaded or has the wrong magnetic tape loaded, or if $MOUNT tries to read a magnetic tape that is not loaded, it notifies the operator to load the correct magnetic tape. $MOUNT will dismount and unload volumes as soon as they have been read or written. The operator can load the next volume in the volume set before the current reel of the volume set reaches the end of the magnetic tape. If writing to the volume set, $MOUNT automatically (1) switches to the next magnetic tape drive; (2) initializes that magnetic tape with the same volume name and protection as specified in the volume labels of the first volume in the set; and (3) notifies the operator that the switch has occurred. If reading the volume set, $MOUNT generates the label for the next volume in the volume set and reads that volume.

Option	Description
	The label name that $MOUNT generates for each additional volume in the volume set consists of six characters: the first four characters are the same as the first four characters of the label name of the previous volume; the fifth and sixth characters represent the number of the volume in the volume set. MNT$M_INIT_CONT applies only to magnetic tapes.
MNT$M_CLUSTER	The volume is to be mounted for clusterwide access; that is, every node on the cluster can access the volume. $MOUNT mounts the volume first on the caller's node and then on every other node in the existing VAXcluster. Only system or group volumes can be mounted clusterwide. If you do not specify MNT$M_GROUP or MNT$M_SYSTEM, $MOUNT mounts the volume as a system volume, provided the caller has SYSNAM privilege. To mount a group volume clusterwide, the caller must have GRPNAM privilege. To mount a system volume clusterwide, the caller must have SYSNAM privilege. MNT$M_CLUSTER has no effect if the system is not a member of a VAXcluster. MNT$M_CLUSTER applies only to disks.

Option	Description
MNT$M_OVR_VOLO	The volume label's **owner identifier** field is not to be processed. $MOUNT reads volume owner and protection information from the **volume owner** field of the volume labels. VMS requires that you specify MNT$M_OVR_VOLO to process magnetic tapes when all of the following conditions exist: (1) the volume was created on a DIGITAL operating system other than VMS; (2) the volume was initialized with a protection specified; and (3) the volume conforms to the Version 3 ANSI label standard. To specify MNT$M_OVR_VOLO, the caller must either have VOLPRO privilege or own the volume. MNT$M_OVR_VOLO applies only to tapes.
MNT$M_TAPE_DATA_WRITE	Enables the tape controller's write cache for this device. Enabling the write cache improves data throughput for write operations. By default, the tape controller's write cache is disabled for the device. This option applies only to tape systems that support a write cache.

MNT$_LIMIT

The MNT$_LIMIT item code specifies the maximum amount of free space in the extent cache. The buffer must contain a longword value, which specifies the amount of free space in units of tenths of a percent of the disk's total free space. The MNT$_LIMIT item code applies only to disks.

MNT$_LOGNAM

The MNT$_LOGNAM item code specifies a logical name for the volume; this logical name is equated to the device name specified by the first MNT$_DEVNAM item code. The buffer must contain a character string from 1 to 64 characters, which is the logical name.

Unless you specify MNT$M_GROUP or MNT$M_SYSTEM, the logical name is entered in the process logical name table.

MNT$_OWNER

The MNT$_OWNER item code specifies the UIC to be assigned ownership of the volume. The buffer must contain a longword octal value, which is the UIC. If the volume is Files-11 structured, the specified value overrides the ownership recorded on the volume. You need either VOLPRO privilege or ownership of the volume to assign a UIC to a Files-11 structured volume.

MNT$_PROCESSOR

For magnetic tapes and Files-11 Structure Level 1 disks, MNT$_PROCESSOR specifies the name of the ancillary control process (ACP) that is to process the volume. The specified ACP overrides the default ACP associated with the device.

For Files–11 Structure Level 2 disks, MNT$_PROCESSOR controls block cache allocation.

To specify MNT$_PROCESSOR, the caller must have OPER privilege.

The buffer must contain a character string specifying either the string UNIQUE, a device name, or a file specification. Following is a description of the action taken for each of these cases.

String	Description
UNIQUE	For magnetic tapes and Files–11 Structure Level 1 disks, **UNIQUE** specifies that $MOUNT create a new process to execute a copy of the default ACP image associated with the device specified by the MNT$_DEVNAM item code.
	For Files–11 Structure Level 2 disks, **UNIQUE** allocates a separate block cache.
ddcu	For magnetic tapes and Files–11 Structure Level 1 disks, **ddcu** specifies that $MOUNT use the ACP process currently being used by the device **ddcu**. The device specified must be in the format ddcu, for example, DRA3.
	For Files–11 Structure Level 1 disks, **ddcu** specifies that $MOUNT take the block allocation from the specified device.
file-spec	Specifies that $MOUNT create a new process to execute the ACP image with the file specification **file-spec**. Wildcard characters are not allowed in the file specification. The file must be in the disk and directory specified by the logical name SYS$SYSTEM. This operation requires CMKRNL privilege.

MNT$_QUOTA

The MNT$_QUOTA item code specifies the size of the quota record cache in units of quota records. The buffer must contain a longword value, which is this size. To specify MNT$_QUOTA, you need OPER privilege. The value 0 disables caching. The MNT$_QUOTA item code applies only to disks.

MNT$_RECORDSIZ

The MNT$_RECORDSIZ item code specifies the number of characters in each record, and is used with MNT$_BLOCKSIZE to specify the data formats for foreign volumes. The buffer must contain a longword value less than or equal to the block size. The MNT$_RECORDSIZ item code applies only to tapes.

If you do not specify MNT$_RECORDSIZ, the record size is assumed to be equal to the block size.

MNT$_SHAMEM

The MNT$_SHAMEM item code applies only to the volume shadowing option. See the *VAX Volume Shadowing Manual.*

MNT$_SHAMEM_COPY

The MNT$_SHAMEM_COPY item code applies only to the volume shadowing option. See the *VAX Volume Shadowing Manual.*

MNT$_SHAMEM_MGCOPY

The MNT$_SHAMEM_MGCOPY item code applies only to the volume shadowing option. See the *VAX Volume Shadowing Manual.*

MNT$_SHANAM

The MNT$_SHANAM item code applies only to the volume shadowing option. See the *VAX Volume Shadowing Manual.*

MNT$_VOLNAM

The MNT$_VOLNAM item code specifies the name of the volume to be mounted on the device. The buffer must contain a character string from 1 to 12 characters, which is the volume name.

The MNT$_VOLNAM item code may appear more than once in an item list; it appears more than once when a volume set is being mounted because, in this case, one volume name is given to each volume in the volume set.

When a disk volume set is being mounted, you must specify MNT$_DEVNAM and MNT$_VOLNAM once for each volume of the volume set. The $MOUNT service mounts the volume specified by the first MNT$_VOLNAM item code on the device specified by the first MNT$_DEVNAM item code in the item list; it mounts the volume specified by the second MNT$_VOLNAM code on the device specified by the second MNT$_DEVNAM code, and so on for all specified volumes and devices. Thus, there must be an equal number of these two item codes in the item list.

When a tape volume set is being mounted, the number of MNT$_DEVNAM item codes specified need not be equal to the number of MNT$_VOLNAM item codes specified, because more than one volume may be mounted on the same device.

MNT$_VOLSET

The MNT$_VOLSET item code specifies the name of a volume set. The buffer must contain a character string from 1 to 12 alphanumeric characters, which is the volume set name.

When you specify MNT$_VOLSET, volumes specified by the MNT$_VOLNAM item code are bound into a new volume set or added to an existing volume set, depending on whether the name specified by MNT$_VOLSET is a new or already existing name.

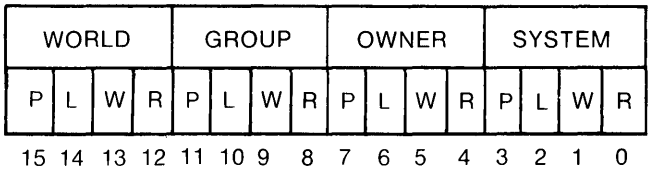
When you specify MNT$_VOLSET to add volumes to an existing volume set, the root volume (RVN1) must either (1) already be mounted or (2) be specified first (by the MNT$_DEVNAM and MNT$_VOLNAM item codes) in the item list.

When you specify MNT$_VOLSET to create a new volume set, the first volume specified (by the MNT$_DEVNAM and MNT$_VOLNAM item codes) in the item list becomes the root volume.

MNT$_VPROT

The MNT$_VPROT item code specifies the protection to be assigned to the volume. The buffer must contain a longword protection mask, which specifies the four types of access allowed to the four categories of user.

The protection mask consists of four 4-bit fields. Each field grants or denies read, write, logical, and physical access to a particular category of user. Cleared bits grant access; set bits deny access. The following diagram depicts the structure of the protection mask.



ZK-1715-84

If you do not specify MNT$_VPROT or specify it as 0, the volume receives the protection that it was assigned when it was initialized. To specify MNT$_VPROT for a Files-11 structured volume, the caller must either own the volume or have VOLPRO privilege.

MNT$_WINDOW

The MNT$_WINDOW item code specifies the number of mapping pointers to be allocated for file windows. The buffer must contain a longword value in the range 7 to 80. This value overrides the default value that was applied when the volume was initialized. The MNT$_WINDOW item code applies only to disks.

When a file is opened, the file system uses the mapping pointers to access the data in the file. To specify MNT$_WINDOW, you need OPER privilege.

DESCRIPTION

To mount a particular volume, the caller must either own or have privilege to access the specified volume or volumes. The privileges required depend on the operation and are listed with the item codes that specify the operation.

The calling process must have TMPMBX or PRMMBX privilege to perform an operator-assisted mount.

When a subprocess mounts a private volume without explicitly allocating the device, the master process of the job becomes the owner of this device. This provision is necessary because the subprocess may be deleted and the volume should remain privately mounted for this job.

When a subprocess explicitly allocates a device and then mounts a private volume on this device, this subprocess retains the device ownership. In this case, only subprocesses of the device owner, and processes with SHARE privilege, have access to the device.

The $MOUNT service uses the following system resources to mount volumes with group or systemwide access allowed:

- Nonpaged pool
- Paged pool

When $MOUNT mounts a disk volume, the logical name DISK$volume-label is always created. If you specify a logical name in the mount request that is different from DISK$volume-label, there will be two logical names associated with the device.

If the logical name of a volume is in a process-private table, then the name is not deleted when the volume is dismounted.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The item list or an address specified in the item list cannot be accessed.
SS$_BADPARAM	A buffer length of zero was specified with a nonzero item code; an illegal item code was specified; or no device was specified.
SS$_NOGRPNAM	The caller does not have GRPNAM privilege.
SS$_NOSYSNAM	The caller does not have SYSNAM privilege.
SS$_NOOPER	The caller does not have the required OPER privilege.
SS$_NOPRIV	The caller does not have sufficient privilege to access a specified volume.
SS$_NOSUCHDEV	The specified device does not exist on the host system.

The $MOUNT service may also return a condition value that is specific to the Mount Utility. The symbolic definition macro $MOUNDEF defines these condition values. For information about how to obtain these symbolic codes, see the *Introduction to VMS System Services*.

$MTACCESS Magnetic Tape Accessibility

The Magnetic Tape Accessibility service allows installations to provide their own routine to interpret and output the accessibility field in the VOL1 and HDR1 labels of an ANSI labeled magnetic tape. (ANSI refers to the *American National Standard for Magnetic Tape Labels and File Structure for Information Interchange*—ANSI, x3.27-1978.)

FORMAT

SYS$MTACCESS *lblnam ,[uic] ,[std_version] ,[access_char] ,[access_spec] ,type*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. For the input of a label, condition values that this service returns are listed under CONDITION VALUES RETURNED. For the output of a label, the value returned in the low byte in R0 is the **access_char** to write to the label.

ARGUMENTS

lblnam

VMS usage: **address**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

ANSI label to be processed. The **lblnam** argument is the address of a longword containing the label. On input, the label passed is either the VOL1 or HDR1 label read from the magnetic tape; on output of labels, this field is zero. The type of label passed is determined by **type**.

uic

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

UIC of the user performing the operation. The **uic** argument is a longword containing the UIC.

std_version

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

Decimal equivalent of the ANSI standard version read from the VOL1 label. The **std_version** argument is a longword containing the standard version number.

access_char

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

Accessibility character specified by the user. The **access_char** argument is a byte containing the accessibility character used for the output of labels.

access_spec

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

Value specifying whether the accessibility character passed in **access_char** was specified by the user. The **access_spec** argument is a byte containing one of the following values.

Value	Meaning
MTA$K_CHARVALID	Yes
MTA$K_NOCHAR	No

This argument is used only for the output of labels.

type

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

Type of accessibility field to process. The **type** argument is a byte containing one of the following values.

Value	Meaning
MTA$K_INVOL1	Input a VOL1 label
MTA$K_INHDR1	Input a HDR1 label
MTA$K_OUTVOL1	Output a VOL1 label
MTA$K_OUTHDR1	Output a HDR1 label

DESCRIPTION

The $MTACCESS service allows installations to provide their own routine to interpret and output the accessibility field in the VOL1 and HDR1 labels of ANSI labeled magnetic tapes. The installation can override the default routine by providing an MTACCESS.EXE executive loaded image. See the *Introduction to VMS System Services* for the procedure for loading an installation-specific executive loaded image.

The default installation routine first checks the ANSI standard version of the label. For magnetic tapes with a version number of 3 or less, the routine outputs either a blank or the character you specified. On input of these magnetic tapes, the routine checks for a blank and returns the value SS$_FILACCERR if the field is not blank.

For magnetic tapes with a version number greater than 3, the routine outputs either the character specified by **access_char** or an ASCII 1 if no character was specified. On input of these magnetic tapes, the routine checks for a blank. If the field is blank, R0 is set to 0. In that case, you are given full access and VMS protection is not checked. If the field contains an ASCII 1 and the VOL1 IMPLEMENTATION IDENTIFIER field contains the VMS system code, R0 is set to SS$_NORMAL. In that case, the VMS protection is checked.

If the field is not blank and does not contain an ASCII 1, R0 is set to SS$_FILACCERR, which forces you to override the accessibility checking, and allows the magnetic tape file system to check VMS protection.

The following summarizes the results of label input check:

Contents of R0	Result
SS$_NORMAL	Check the VMS protection on the magnetic tape.
0	Give the user full access. VMS protection is not checked.
SS$_FILACCERR	Check for explicit override, then check VMS protection.

Note that the default accessibility routine does not output SS$_NOVOLACC or SS$_NOFILACC. These statuses are included for the installation's use, and the magnetic file system handles these cases.

The magnetic tape file system calls $MTACCESS to process the accessibility field in the VOL1 and HDR1 labels. After a call to the system service, the magnetic tape file system checks to ensure that the installation did not move the magnetic tape. If the magnetic tape was moved, the magnetic tape file system completes the current operation with an SS$_TAPEPOSLOST error. Finally, it processes the remainder of the label according to the status returned by $MTACCESS.

Because accessibility is an installation-provided routine, VMS cannot determine which users have the authority to override the processing of this field. However, the magnetic tape file system allows only operator class users to deal with blank magnetic tapes so that a user must have both OPER and VOLPRO privileges to initialize or mount blank magnetic tapes.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_FILACCERR	The accessibility characteristic in the HDR1 label is not blank and you cannot access the file without overriding the field.
SS$_NOFILACC	The user has no access to the file.
SS$_NOVOLACC	The user has no access to the volume.

$NUMTIM Convert Binary Time to Numeric Time

The Convert Binary Time to Numeric Time service converts an absolute or delta time from 64-bit system time format to binary integer date and time values.

FORMAT

SYS$NUMTIM *timbuf ,[timadr]*

RETURNS

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

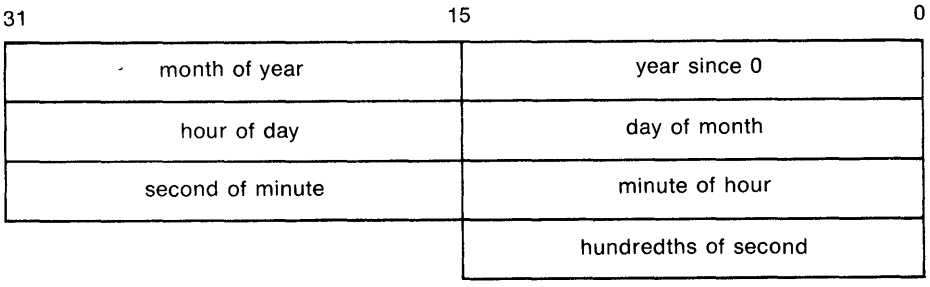
Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

timbuf

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

Buffer into which $NUMTIM writes the converted date and time. The **numtim** argument is the address of a 7-word structure. The following diagram depicts the fields in this structure.



ZK-1716-84

If the **timadr** argument specifies a delta time, $NUMTIM returns 0 in the **year since 0** and **month of year** fields. It returns in the **day of month** field the number of days specified by the delta time, which must be less than 10,000 days.

timadr

VMS usage: **date_time**
type: **quadword (unsigned)**
access: **read only**
mechanism: **by reference**

The 64-bit time value to be converted. The **timadr** argument is the address of a quadword containing this time. A positive time value represents an absolute time, while a negative time value indicates a delta time.

If you do not specify **timadr**, $NUMTIM returns the current system time.

If **timadr** specifies 0, $NUMTIM returns the base date (November 17, 1858).

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The 64-bit time value cannot be read by the caller, or the buffer cannot be written by the caller.
SS$_IVTIME	The specified delta time is equal to or greater than 10,000 days.

$PARSE_ACL Parse Access Control List Entry

The Parse Access Control List Entry service parses the specified text string and converts it into the binary representation for an access control list entry (ACE).

FORMAT

SYS$PARSE_ACL *aclstr ,aclent ,[errpos] ,[accnam] ,[nullarg] ,[nullarg]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

aclstr

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

Formatted ACE that is parsed when $PARSE_ACL completes execution. The **aclstr** argument is the address of a string descriptor pointing to the text string to be parsed.

aclent

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

Description of the ACE that is parsed when $PARSE_ACL completes execution. The **aclent** argument is the address of a descriptor pointing to the buffer in which the ACE is written. The first byte of the buffer contains the length of the ACE; the second byte contains a value that identifies the type of ACE, which in turn defines the format of the ACE. For information about the ACE types and their associated formats, see the SYS$FORMAT_ACL service.

errpos

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

Number of characters from **aclstr** processed by SYS$PARSE_ACL. The **errpos** argument is the address of a word that receives the number of

characters actually processed by the service. If the service fails, this count points to the failing point in the string.

accnam

VMS usage: **access_bit_names**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Names of the bits in the access mask when $PARSE_ACL is executing. The **accnam** argument is the address of an array of 32 quadword descriptors that define the names of the bits in the access mask. Each element points to the name of a bit. The first element names bit 0, the second element names bit 1, and so on. If you omit **accnam**, the following names are used:

```
Bit 0    READ
Bit 1    WRITE
Bit 2    EXECUTE
Bit 3    DELETE
Bit 4    CONTROL
Bit 5    BIT_5
Bit 6    BIT_6
   .
   .
   .
Bit 31   BIT_31
```

nullarg

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

Place-holding argument reserved by DIGITAL.

DESCRIPTION

The Parse Access Control List Entry service parses the specified text string and converts it into the binary representation for an access control list entry.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The string or its descriptor cannot be read by the caller, or the buffer descriptor cannot be read by the caller, or the buffer cannot be written by the caller, or the buffer is too small to hold the ACL entry.
SS$_IVACL	The format of the access control list entry is not valid.

$PURGWS Purge Working Set

The Purge Working Set service removes a specified range of pages from the current working set of the calling process to make room for pages required by a new program segment. However, the Adjust Working Set Limit ($ADJWSL) service is the preferred mechanism for controlling a process's use of physical memory resources.

FORMAT

SYS$PURGWS *inadr*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENT

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

Starting and ending virtual addresses of the range of pages to be purged. The **inadr** argument is the address of a 2-longword array containing, in order, the starting and ending process virtual addresses. Only the virtual page number portion of each virtual address is used; the low-order 9 bits are ignored.

The $PURGWS service locates pages within the specified range and removes them if they are in the working set.

If the starting and ending virtual addresses are the same, only that single page is purged.

To purge the entire working set, specify a range of pages from 0 through 7FFFFFFF; in this case, the image will continue to execute and pages will be faulted back into the working set as they are needed.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The input address array cannot be read by the caller.

$PUTMSG Put Message

The Put Message service writes one or more error messages to SYS$ERROR (and to SYS$OUTPUT if it is different from SYS$ERROR). The $PUTMSG service is a generalized message-formatting and output routine used by VMS to write informational and error messages to processes.

FORMAT

SYS$PUTMSG *msgvec ,[actrtn] ,[facnam] ,[actprm]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

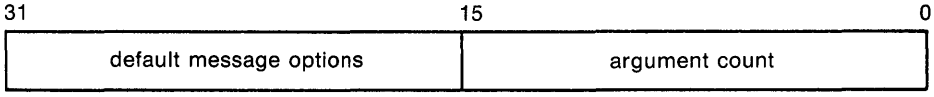
ARGUMENTS

msgvec

VMS usage: **cntrlblk**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Message argument vector specifying the message or messages to be written and options that $PUTMSG is to use in writing the message or messages. The **msgvec** argument is the address of the message vector.

The message vector consists of one longword followed by one or more message descriptors, one descriptor per message. The following diagram depicts the contents of the first longword.



ZK-1717-84

Message Vector Fields

argument count
This word-length field specifies the total number of longwords in the message vector, not including the first longword (of which it is a part).

default message options
This word-length field specifies which message component or components are to be written. The **default message options** field is a word-length bit vector wherein a bit, when set, specifies that the corresponding message component is to be written. For a description of each of these components, refer to the DESCRIPTION section.

The following table shows the significant bit numbers. Note that the bit numbers shown (0, 1, 2, 3) are the bit positions from the beginning of the word; however, because the word is the second word in the longword, you should add the number 16 to each bit number to specify its exact offset within the longword.

Bit	Value	Description
0	1	Include message text
	0	Do not include message text
1	1	Include mnemonic name for message text
	0	Do not include mnemonic name for message text
2	1	Include severity level indicator
	0	Do not include severity level indicator
3	1	Include facility prefix
	0	Do not include facility prefix

Bits 4 through 15 must be 0.

You can override the default setting specified by the **default message options** field for any or all messages by specifying different options in the **new message options** field of any subsequent message descriptor. When you specify **new message options**, the options it specifies become the new default settings for all remaining messages until you specify **new message options** again.

The $PUTMSG service passes the **default message flags** field to the $GETMSG service as the **flags** argument.

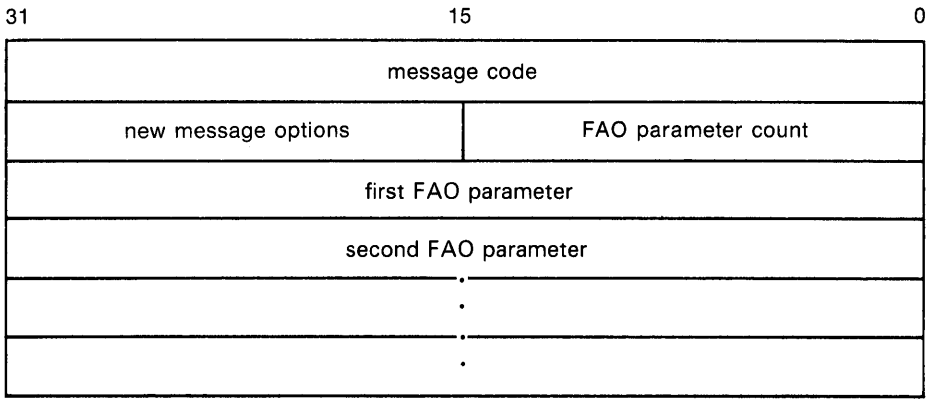
If you do not specify the **default message flags** field, the default message options for the process are used; you can set the process default message options by using the DCL command SET MESSAGE.

The DESCRIPTION section shows the format that $PUTMSG uses to write these message components.

Following the first longword of the message vector are one or more message descriptors. A message descriptor may have one of four possible formats, depending on the type of message it describes. There are four types of message:

- User-supplied
- System
- VMS RMS
- System exception

Message Descriptor for User-Supplied Messages



ZK-1718-84

Fields in Message Descriptor for User-Supplied Messages

message code

Longword value that uniquely identifies the message. The DESCRIPTION section discusses the message code; the *VMS Message Utility Manual* explains how to create message codes.

FAO parameter count

Word-length value specifying the number of longword FAO parameters that follow in the message descriptor. The number of FAO parameters needed depends on the FAO directives used in the message text; some FAO directives require one or more parameters, while some directives require none.

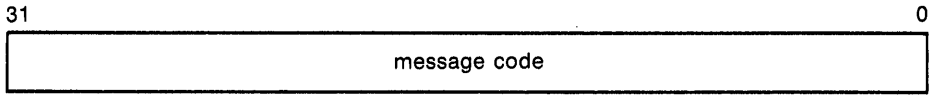
new message options

Word-length bit vector specifying new message options for the current message. The contents and format of this field is identical to that of the **default message options** field.

FAO parameter

Longword value used by an FAO directive appearing in the message text. The FAO parameters listed in the message descriptor must appear in the order in which they will be used by the FAO directives in the message text.

Message Descriptor for System Messages



ZK-1719-84

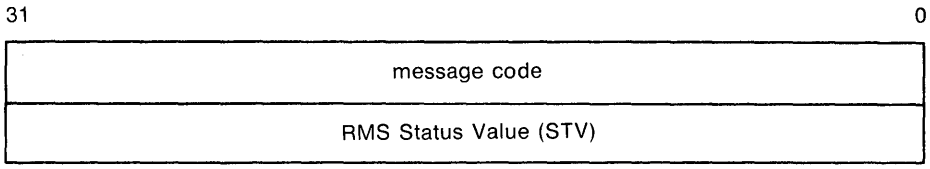
Fields in Message Descriptor for System Messages

message code

Longword value that uniquely identifies the message. The **facility number** field in the message code identifies the facility associated with the message. A system message has a facility number of 0. You cannot specify the **FAO**

parameter count, **new message options**, and **FAO parameter** fields. Each longword following the **message identification** field in the message vector will be interpreted as another message identification.

Message Descriptor for VMS RMS Messages



ZK-1720-84

Fields in Message Descriptor for VMS RMS Messages

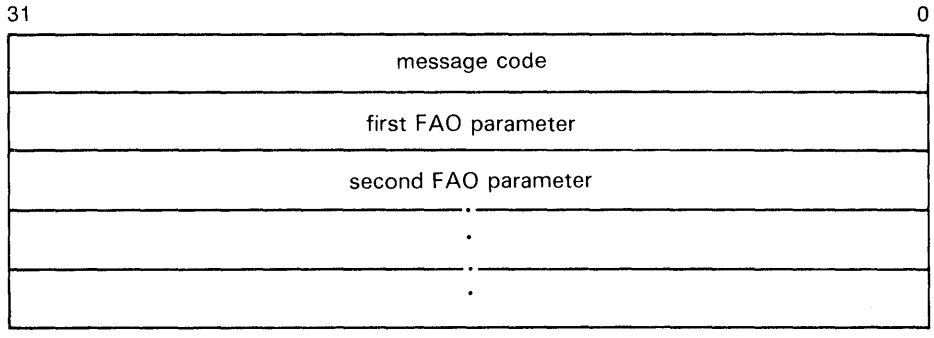
message code

Longword value that uniquely identifies the message. The **facility number** field in the message code identifies the facility associated with the message. An RMS message has a facility number of 1. You cannot specify the **FAO parameter count**, **new message options**, and **FAO parameter** fields. The longword following the **message identification** field in the message vector will be interpreted as a standard value field (STV).

RMS Status Value

Longword containing an STV for use by an RMS message that has an associated STV value. The $PUTMSG service uses the STV value as an FAO parameter or as another message identification, depending on the RMS message identified by the **message identification** field. If the RMS message does not have an associated STV, $PUTMSG ignores the STV longword in the message descriptor.

Message Descriptor for System Exception Messages



ZK-1721-84

Fields in Message Descriptor for System Exception Messages

message code

Longword value that uniquely identifies the message. The facility number field in the message code identifies the facility associated with the message. A system exception message has a facility number of 0. You cannot specify

the **FAO parameter count** and **new message options** fields. The longword or longwords following the **message code** field in the message vector will be interpreted as FAO parameters.

actrtn

VMS usage: **procedure**
type: **procedure entry mask**
access: **call without stack unwinding**
mechanism: **by reference**

User-supplied action routine to be executed during message processing. The **actrtn** argument is the address of the entry mask of this routine.

Note that the first argument passed to the action routine is the address of a character string descriptor pointing to the message text; the parameter specified by **actprm** is the second.

The action routine receives control after a message is formatted but before it is actually written to the user.

The completion code in general register R0 from the action routine indicates whether the message should be written. If the low-order bit of R0 is set (1), then the message will be written. If the low-order bit is cleared (0), then the message will not be written.

If you do not specify **actrtn** or you specify it as 0 (the default), no action routine executes.

Because $PUTMSG writes messages only to SYS$ERROR and SYS$OUTPUT, an action routine is useful when output must be directed to, for example, a file.

facnam

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

Facility prefix to be used in the first or only message written by $PUTMSG. The **facnam** argument is the address of a character string descriptor pointing to this facility prefix.

If you do not specify **facnam**, $PUTMSG uses the default facility prefix associated with the message.

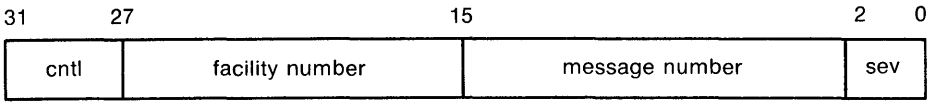
actprm

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

Parameter to be passed to the action routine. The **actprm** argument is a longword value containing this parameter. If you do not specify **actprm**, no parameter is passed.

DESCRIPTION

In VMS, a message is identified by a longword value, which is called the **message code**. To construct a message code, you specify values for its four fields, using the Message Utility. The following diagram depicts the longword message code.



ZK-1722-84

Thus, each message has a unique longword value associated with it: its message code. You can give this longword value a symbolic name using the Message Utility. Such a symbolic name is called the **message symbol**.

The Message Utility describes how to construct a message symbol according to the conventions for VMS messages. Basically, the message symbol has two parts: (1) a facility prefix, which is an abbreviation of the name of the facility with which the message is associated, and (2) a mnemonic name for the message text, which serves to hint at the nature of the message. These two parts are separated by an underscore character (_) in the case of a user-constructed message and by a dollar sign/underscore ($_) in the case of system messages.

The message components written by $PUTMSG are derived both from the message code and from the message symbol. For additional information about both the message code and the message symbol, refer to the *VMS Message Utility Manual*.

The $PUTMSG service writes the message components in the following format:

%FACILITY-L-IDENT, message text

where:

%	Is the prefix used for the first message written. The hyphen (-) is the prefix used for the remaining messages.
FACILITY	Is the facility prefix taken from the message symbol. This facility prefix may be overridden by a facility prefix specified in the **facnam** argument in the call to $PUTMSG.
L	Is the severity level indicator. The severity level indicator is taken from the message code.
IDENT	Is a mnemonic name for the message text, taken from the message symbol.
message text	Is the message text specified in the message source file.

The $PUTMSG service does not check the length of the argument list and therefore cannot return the SS$_INSFARG (insufficient arguments) condition value. Be sure you specify the required number of arguments.

If an error occurs while $PUTMSG calls the Formatted ASCII Output ($FAO) service, FAO parameters specified in the message vector do not appear in the output.

You cannot call the $PUTMSG service from kernel mode.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.

EXAMPLES

1

```
VECTOR: .LONG   3               ; Argument count & null msg. flags
        .LONG   SS$_ABORT       ; Abort message
        .LONG   RMS$_FNF        ; File not found message
        .LONG   0               ; Null STV parameter
         .
         .
         .
        $PUTMSG_S -
                MSGVEC=VECTOR
```

This example shows a segment of a program used to request $PUTMSG to write the following messages to the current SYS$OUTPUT device (and to SYS$ERROR, if it is different):

- The complete message associated with the system status code SS$_ABORT (%SYSTEM-F-ABORT, abort)
- The complete message associated with the system status code RMS$_FNF (-RMS-E-FNF, file not found)

2

```
INTEGER STATUS,
2       OLDHND

CHARACTER*5 NUM

INCLUDE '($SSDEF)'
INCLUDE '($LIBDEF)'

INTEGER LIB$GET_INPUT,
2       LIB$ESTABLISH,
2       SYS$GETJPI
EXTERNAL ERR

OPEN (UNIT = 1,
2     TYPE = 'NEW',
2     CARRIAGECONTROL = 'LIST',
2     FILE = 'ERROR.LOG')

OLDHND = LIB$ESTABLISH (ERR)

! This routine executes successfully
STATUS = LIB$GET_INPUT (NUM, 'NUM: ')
IF (.NOT. STATUS) CALL LIB$SIGNAL (%VAL(STATUS))
! This routine fails with insufficient arguments
STATUS = SYS$GETJPI(,)
IF (.NOT. STATUS) CALL LIB$SIGNAL (%VAL(STATUS))

END
```

```
INTEGER FUNCTION ERR (SIGARGS,
2                     MECHARGS)

INTEGER SIGARGS(*),
2       MECHARGS(*)
INTEGER NEWSIGARGS(10), ! Must specify a length for
                        ! array so choose one large enough
                        ! to cover any eventuality
2       ELEMENT

INCLUDE '($SSDEF)'

EXTERNAL PUT_LINE
INTEGER PUT_LINE

! Get rid of last two elements in SIGARGS (the PC and PSL),
! then pad NEWSIGARGS with zeros.

ELEMENT = 1
NEWSIGARGS(ELEMENT) = 10

DO I = 1, SIGARGS(1) - 2
  ELEMENT = ELEMENT + 1
  NEWSIGARGS (ELEMENT) = SIGARGS (ELEMENT)
  END DO
DO I = ELEMENT + 1, 10
  ELEMENT = ELEMENT + 1
  NEWSIGARGS (ELEMENT) = 0
  END DO

CALL SYS$PUTMSG (NEWSIGARGS, PUT_LINE,)
ERR = SS$_RESIGNAL
                     ! Could use CONTINUE and let $PUTMSG
                     ! write the message

END
INTEGER FUNCTION PUT_LINE (LINE)

CHARACTER*(*) LINE

PUT_LINE = 0         ! Since you're resignalling, don't let
                     ! SYS$PUTMSG write the error.

WRITE (UNIT = 1,
2      FMT = '(A)') LINE
END
```

This VAX FORTRAN example uses $PUTMSG to write any error messages to a file (ERROR.LOG) as well as to the terminal.

$QIO Queue I/O Request

The Queue I/O Request service queues an I/O request to a channel associated with a device.

The $QIO service completes asynchronously; that is, it returns to the caller immediately after queuing the I/O request, without waiting for the I/O operation to complete.

For synchronous completion, you use the Queue I/O Request and Wait ($QIOW) service. The $QIOW service is identical to the $QIO service in every way except that $QIOW returns to the caller after the I/O operation has completed.

For additional information about system service completion, refer to the Synchronize ($SYNCH) service and to the *Introduction to VMS System Services*.

FORMAT

SYS$QIO *[efn] ,chan ,func [,iosb] [,astadr] [,astprm] [,p1] [,p2] [,p3] [,p4] [,p5] [,p6]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

efn

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

Event flag that $QIO is to set when the I/O operation completes. The **efn** argument is a longword value containing the number of the event flag; however, $QIO uses only the low-order byte.

If you do not specify **efn**, event flag 0 is set.

When $QIO begins execution, it clears the specified event flag or event flag 0 if **efn** was not specified.

The specified event flag is set if the service terminates without queuing an I/O request.

chan

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

I/O channel assigned to the device to which the request is directed. The **chan** argument is a longword value containing the number of the I/O channel; however, $QIO uses only the low-order word.

func

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

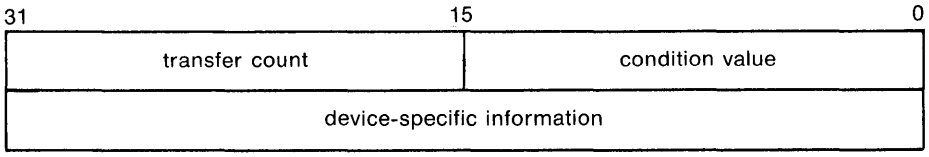
Device-specific function codes and function modifiers specifying the operation to be performed. The **func** argument is a longword value containing the function code.

Each device has its own function codes and function modifiers. For complete information about the function codes and function modifiers that apply to the particular device to which the I/O operation is to be directed, refer to the *VMS I/O User's Reference Volume.*

iosb

VMS usage: **io_status_block**
type: **quadword (unsigned)**
access: **write only**
mechanism: **by reference**

I/O status block to receive the final completion status of the I/O operation. The **iosb** argument is the address of the quadword I/O status block. The following diagram depicts the structure of the I/O status block.



ZK-1723-84

I/O Status Block Fields

condition value
Word-length condition value that $QIO returns when the I/O operation actually completes.

transfer count
Number of bytes of data transferred in the I/O operation. For information about how specific devices handle this field of the I/O status block, refer to the *VMS I/O User's Reference Volume.*

device-specific information
Contents of this field vary depending on the specific device and on the specified function code. For information on how specific devices handle this field of the I/O status block, refer to the *VMS I/O User's Reference Volume.*

When $QIO begins execution, it clears the quadword I/O status block if the **iosb** argument is specified.

Though this argument is optional, DIGITAL strongly recommends that you specify it, for the following reasons:

- If you are using an event flag to signal the completion of the service, you can test the I/O status block for a condition value to be sure that the event flag was not set by an event other than service completion.
- If you are using the $SYNCH service to synchronize completion of the service, the I/O status block is a required argument for $SYNCH.
- The condition value returned in R0 and the condition value returned in the I/O status block provide information about different aspects of the call to the $QIO service. The condition value returned in R0 gives you information about the success or failure of the service call itself; the condition value returned in the I/O status block gives you information about the success or failure of the service operation. Therefore, to accurately assess the success or failure of the call to $QIO, you must check the condition values returned in both R0 and the I/O status block.

astadr

VMS usage: **ast_procedure**
type: **procedure entry mask**
access: **call without stack unwinding**
mechanism: **by reference**

AST service routine to be executed when the I/O completes. The **astadr** argument is the address of a longword value that is the entry mask to the AST routine.

The AST routine executes at the access mode of the caller of $QIO.

astprm

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

AST parameter to be passed to the AST service routine. The **astprm** argument is a longword value containing the AST parameter.

p1 to p6

VMS usage: **varying_arg**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference or by value depending on the I/O function**

Optional device- and function-specific I/O request parameters.

For more information about these parameters, see the *VMS I/O User's Reference Volume*.

DESCRIPTION

The $QIO service operates only on assigned I/O channels and only from access modes that are equal to or more privileged than the access mode from which the original channel assignment was made.

The $QIO service uses the following system resources:

- The process's quota for buffered I/O limit (BIOLM) or direct I/O limit (DIOLM)

- The process's buffered I/O byte count (BYTLM) quota
- The process's AST limit (ASTLM) quota, if an AST service routine is specified
- System dynamic memory to construct a database to queue the I/O request
- Possibly, additional memory on a device-dependent basis

For $QIO, you can synchronize completion by (1) specifying the **astadr** argument to have an AST routine execute when the I/O completes or (2) by calling the Synchronize ($SYNCH) service to await completion of the I/O operation. The $QIOW service completes synchronously, and it is the best choice when synchronous completion is required.

For information about how to use the $QIO service for network operations, refer to the *VMS Networking Manual*.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully. The I/O request was successfully queued.
SS$_ABORT	A network logical link was broken.
SS$_ACCVIO	Either the I/O status block cannot be written by the caller, or the parameters for device-dependent function codes are specified incorrectly.
SS$_DEVOFFLINE	The specified device is off line and not currently available for use.
SS$_EXQUOTA	The process has (1) exceeded its AST limit (ASTLM) quota, (2) exceeded its buffered I/O byte count (BYTLM) quota, (3) exceeded its buffered I/O limit (BIOLM) quota, (4) exceeded its direct I/O limit (DIOLM) quota, or (5) requested a buffered I/O transfer smaller than the buffered byte count quota limit (BYTLM), but when added to other current buffer requests, the buffered I/O byte count quota was exceeded.
SS$_ILLEFC	You specified an illegal event flag number.
SS$_INSFMEM	The system dynamic memory is insufficient for completing the service.
SS$_IVCHAN	You specified an invalid channel number, that is, a channel number of 0 or a number larger than the number of channels available.
SS$_NOPRIV	The specified channel does not exist, was assigned from a more privileged access mode, or the process does not have the necessary privileges to perform the specified functions on the device associated with the specified channel.
SS$_UNASEFC	The process is not associated with the cluster containing the specified event flag.
SS$_LINKABORT	The network partner task aborted the logical link.

SS$_LINKDISCON	The network partner task disconnected the logical link.
SS$_PATHLOST	The path to the network partner task node was lost.
SS$_PROTOCOL	A network protocol error occurred. This is most likely due to a network software error.
SS$_CONNECFAIL	The connection to a network object timed out or failed.
SS$_FILALRACC	A logical link is already accessed on the channel (that is, a previous connect on the channel).
SS$_INVLOGIN	The access control information was invalid at the remote node.
SS$_IVDEVNAM	The NCB has an invalid format or content.
SS$_LINKEXIT	The network partner task was started, but exited before confirming the logical link (that is, $ASSIGN to SYS$NET).
SS$_NOLINKS	No logical links are available. The maximum number of logical links as set for the executor MAXIMUM LINKS parameter was exceeded.
SS$_NOSUCHNODE	The specified node is unknown.
SS$_NOSUCHOBJ	The network object number is unknown at the remote node; or for a TASK= connect, the named DCL command procedure file cannot be found at the remote node.
SS$_NOSUCHUSER	The remote node could not recognize the login information supplied with the connection request.
SS$_PROTOCOL	A network protocol error occurred. This error is most likely due to a network software error.
SS$_REJECT	The network object rejected the connection.
SS$_REMRSRC	The link could not be established because system resources at the remote node were insufficient.
SS$_SHUT	The local or remote node is no longer accepting connections.
SS$_THIRDPARTY	The logical link was terminated by a third party (for example, the system manager).
SS$_TOOMUCHDATA	The task specified too much optional or interrupt data.
SS$_UNREACHABLE	The remote node is currently unreachable.

CONDITION VALUES RETURNED IN THE I/O STATUS BLOCK

Device-specific condition values; the *VMS I/O User's Reference Volume* lists these condition values for each device.

$QIOW Queue I/O Request and Wait

The Queue I/O Request and Wait service queues an I/O request to a channel associated with a device.

The $QIOW service completes synchronously; that is, it returns to the caller after the I/O operation has actually completed.

For asynchronous completion, you use the Queue I/O Request ($QIO) service; $QIO returns to the caller after queuing the I/O request, without waiting for the I/O operation to be completed.

In all other respects, $QIOW is identical to $QIO. For all other information about the $QIOW service, refer to the documentation of $QIO.

For additional information about system service completion, refer to the Synchronize ($SYNCH) service and to the *Introduction to VMS System Services*.

FORMAT

SYS$QIOW *[efn] ,chan ,func [,iosb] [,astadr] [,astprm] [,p1] [,p2] [,p3] [,p4] [,p5] [,p6]*

$READEF Read Event Flags

The Read Event Flags service returns the current status of all 32 event flags in a local or common event flag cluster. In addition, the condition value returned indicates whether the specified event flag is set or clear.

FORMAT

SYS$READEF *efn ,state*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

efn

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

Number of any event flag in the cluster whose status is to be returned. The **efn** argument is a longword containing this number; however, $READEF uses only the low-order byte. Specifying an event flag within a cluster requests that $READEF return the status of all event flags in that cluster.

There are two local event flag clusters, which are local to the process: cluster 0 and cluster 1. Cluster 0 contains event flag numbers 0 to 31, and cluster 1 contains event flag numbers 32 to 63.

There are two common event flag clusters: cluster 2 and cluster 3. Cluster 2 contains event flag numbers 64 to 95, and cluster 3 contains event flag numbers 96 to 127.

state

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

State of all event flags in the specified cluster. The **state** argument is the address of a longword into which $READEF writes the state (set or clear) of the 32 event flags in the cluster.

CONDITION VALUES RETURNED

SS$_WASCLR	The service completed successfully. The specified event flag is clear.
SS$_WASSET	The service completed successfully. The specified event flag is set.
SS$_ACCVIO	The longword that is to receive the current state of all event flags in the cluster cannot be written by the caller.
SS$_ILLEFC	You specified an illegal event flag number.
SS$_UNASEFC	The process is not associated with the cluster containing the specified event flag.

$REM_HOLDER Remove Holder Record from Rights Database

Deletes the specified holder record from the target identifier's list of holders.

FORMAT

SYS$REM_HOLDER *id ,holder*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

id

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

Binary value of target identifier whose holder is deleted when $REM_HOLDER completes execution. The **id** argument is a longword containing the identifier value.

holder

VMS usage: **rights_holder**
type: **quadword (unsigned)**
access: **read only**
mechanism: **by reference**

Identifier of holder being deleted when $REM_HOLDER completes execution. The **holder** argument is the address of a quadword containing the UIC identifier of the holder in the first longword and the value of zero in the second longword.

DESCRIPTION

The Remove Holder Record from Rights Database service removes the specified holder record from the target identifier's list of holders.

You need write access to the rights database to use this service. If the database is in SYS$SYSTEM (the default), you need SYSPRV privilege to grant write access to the database.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The **id** or **holder** argument cannot be read by the caller.
SS$_INSFMEM	The process dynamic memory is insufficient for opening the rights database.
SS$_IVIDENT	The specified identifier or holder identifier is of invalid format.
SS$_NOSUCHID	The specified identifier does not exist in the rights database, or the specified holder identifier does not exist in the rights database.
RMS$_PRV	The user does not have write access to the rights database.

Because the rights database is an indexed file accessed with VMS RMS, this service may also return RMS status codes associated with operations on indexed files. For descriptions of these status codes, refer to the *VMS Record Management Services Manual.*

$REM_IDENT Remove Identifier from Rights Database

The Remove Identifier from Rights Database service removes the specified identifier record and all its holder records (if any) from the rights database.

FORMAT

SYS$REM_IDENT *id*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENT

id

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

Binary value of identifier deleted from rights database when $REM_IDENT completes execution. The **id** argument is a longword containing the identifier value.

DESCRIPTION

The $REM_IDENT system service deletes the specified identifier from the rights database. All holder records associated with the identifier are also deleted. In addition, any records in identifiers that the deleted identifier held are also deleted.

You need write access to the rights database to use this service. If the database is in SYS$SYSTEM (the default), you need SYSPRV privilege to grant write access to the database.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_INSFMEM	The process dynamic memory is insufficient for opening the rights database.
SS$_IVIDENT	The specified identifier is of invalid format.

SS$_NOSUCHID	The specified identifier does not exist in the rights database.
RMS$_PRV	The user does not have write access to the rights database.

Because the rights database is an indexed file accessed with VMS RMS, this service may also return RMS status codes associated with operations on indexed files. For descriptions of these status codes, refer to the *VMS Record Management Services Manual.*

$RESUME Resume Process

The Resume Process service (1) causes a process previously suspended by the Suspend Process ($SUSPND) service to resume execution or (2) cancels the effect of a subsequent suspend request.

FORMAT

SYS$RESUME *[pidadr] ,[prcnam]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

pidadr

VMS usage: **process_id**
type: **longword (unsigned)**
access: **modify**
mechanism: **by reference**

Process identification (PID) of the process to be resumed. The **pidadr** argument is the address of a longword containing the PID.

You must specify the **pidadr** argument to delete processes in other UIC groups.

prcnam

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

Name of the process to be resumed. The **prcnam** argument is the address of a character string descriptor pointing to the process name, which is a character string of from 1 to 15 characters.

You can use the **prcnam** argument to resume only processes in the same UIC group as the calling process, because process names are unique to UIC groups, and VMS uses the UIC group number of the calling process to interpret the process name specified by the **prcnam** argument. You must use the **pidadr** argument to delete processes in other UIC groups.

If you specify neither the **pidadr** nor **prcnam** argument, the resume request is issued on behalf of the calling process.

DESCRIPTION

Depending on the operation, the calling process may need a certain privilege to use $RESUME:

- GROUP privilege to resume execution of a process in the same group unless the process has the same UIC as the calling process
- WORLD privilege to resume execution of any process in the system

If one or more resume requests are issued for a process that is not suspended, a subsequent suspend request completes immediately; that is, the process is not suspended. No count is maintained of outstanding resume requests.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The process name string or string descriptor cannot be read by the caller, or the process identification cannot be written by the caller.
SS$_IVLOGNAM	The specified process name has a length of 0 or has more than 15 characters.
SS$_NONEXPR	The specified process does not exist, or an invalid process identification was specified.
SS$_NOPRIV	The process does not have the privilege to resume the execution of the specified process.

$REVOKID Revoke Identifier from Process

The Revoke Identifier from Process service removes the specified identifier from the rights list of the process or the system. If the identifier is listed as a holder of any other identifier, the appropriate holder records are also deleted.

FORMAT

SYS$REVOKID *[pidadr] ,[prcnam] ,[id] ,[name] ,[prvatr]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

pidadr

VMS usage: **process_id**
type: **longword (unsigned)**
access: **modify**
mechanism: **by reference**

Process identification (PID) number of the process affected when $REVOKID completes execution. The **pidadr** argument is the address of longword containing the PID of the process to be affected. You use –1 to indicate the system rights list. When **pidadr** is passed, it is also returned; therefore, you must pass it as a variable rather than a constant.

prcnam

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

Process name on which $REVOKID operates. The **prcnam** argument is the address of a character string descriptor containing the process name. The maximum length of the name is 15 characters. Because the UIC group number is interpreted as part of the process name, you must use **pidadr** to specify the rights list of a process in a different group.

id

VMS usage: **rights_id**
type: **quadword (unsigned)**
access: **modify**
mechanism: **by reference**

Identifier and attributes to be removed when $REVOKID completes execution. The **id** argument is the address of a quadword containing the binary identifier

code to be removed in the first longword and the attributes in the second longword.

Symbol values are offsets to the bits within the longword. You can also obtain the values as masks with the appropriate bit set using the prefix KGB$M rather than KGB$V. The following symbols for each bit position are defined in the system macro library ($KGBDEF).

Bit Position	Meaning When Set
KGB$V_DYNAMIC	Allows the unprivileged holder to add or remove the identifier from the process rights list.
KGB$V_RESOURCE	Allows the holder to charge resources, such as disk blocks, to the identifier.

You must specify either **id** or **name**. Because the **id** argument is returned as well as passsed if you specify **name**, you must pass it as a variable rather than a constant in this case.

name

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

Name of the identifier removed when $REVOKID completes execution. The **name** argument is the address of a descriptor pointing to the name of the identifier.

prvatr

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

Attributes of the deleted identifier. The **prvatr** argument is the address of a longword used to store the attributes of the identifier.

DESCRIPTION

Because the Revoke Identifier from Process service removes the specified identifier from the rights list of the process or the system, this service is meant for use by a privileged subsystem to alter the access rights profile of a user, based on installation policy. It is not meant for use by the general system user.

You need CMKRNL privilege to invoke this service. In addition, you need GROUP privilege to modify the rights list of a process in the same group as the calling process (unless the process has the same UIC as the calling process). You need WORLD privilege to modify the rights list of a process outside the caller's group. You need SYSNAM privilege to modify the system rights list.

The result of passing the **pidadr** or the **prcnam** argument or both to SYS$REVOKID is summarized in the following table.

prcnam	pidadr	Result
Omitted	Omitted	Current process ID is used; process ID is not returned.
Omitted	0	Current process ID is used; process ID is returned.
Omitted	Specified	Specified process ID is used; process ID is returned.
Specified	Omitted	Specified process name is used; process ID is not returned.
Specified	0	Specified process name is used; process ID is returned.
Specified	Specified	Specified process ID is used; process ID is returned; process name is ignored.

The result of passing either the **name** or the **id** argument or both to SYS$REVOKID is summarized in the following table.

name	id	Result
Omitted	Omitted	Illegal
Omitted	Specified	Specified identifier value is used; identifier value is returned.
Specified	Omitted	Specified identifier name is used; identifier value is not returned.
Specified	0	Specified identifier name is used; identifier value is returned.
Specified	Specified	Specified identifier value is used, identifier value is returned, identifier name is ignored.

CONDITION VALUES RETURNED

SS$_WASCLR	The service completed successfully; the rights list did not contain the specified identifier.
SS$_WASSET	The service completed successfully; the rights list already held the specified identifier.
SS$_ACCVIO	The **pidadr** argument cannot be read or written, or **prcnam** cannot be read, or **id** cannot be read or written, or **name** cannot be read, or **prvatr** cannot be written.
SS$_INSFMEM	The process dynamic memory is insufficient for opening the rights database.
SS$_NOPRIV	The caller does not have CMKRNL privilege; or is not running in exec or kernel mode; or the caller lacks GROUP, WORLD, or SYSNAM privilege as required.
SS$_NOSUCHID	The specified identifier name does not exist in the rights database. Note that the binary identifier, if given, is not validated against the rights database.

SS$_RIGHTSFULL	The rights list of the process or system is full.
SS$_IVIDENT	The specified identifier or holder is of invalid format, or the specified identifier and holder are equal.
SS$_NOSYSNAM	The operation requires SYSNAM privilege.
SS$_IVLOGNAM	You specified an invalid logical name.
SS$_NONEXPR	You specified a nonexistent process.
RMS$_PRV	The user does not have read access to the rights database.

Because the rights database is an indexed file accessed with VMS RMS, this service may also return RMS status codes associated with operations on indexed files. For descriptions of these status codes, refer to the *VMS Record Management Services Manual.*

$SCHDWK Schedule Wakeup

The Schedule Wakeup service schedules the awakening of a process that has placed itself in a state of hibernation with the Hibernate ($HIBER) service. A wakeup can be scheduled for a specified absolute time or for a delta time, and can be repeated at fixed intervals.

FORMAT

SYS$SCHDWK *[pidadr] ,[prcnam] ,daytim ,[reptim]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

pidadr

VMS usage: **process_id**
type: **longword (unsigned)**
access: **modify**
mechanism: **by reference**

Process identification (PID) of the process to be awakened. The **pidadr** argument is the address of a longword containing the PID.

You must specify the **pidadr** argument to awaken processes in other UIC groups.

prcnam

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

Name of the process to be awakened. The **prcnam** is the address of a character string descriptor pointing to the process name, which is a character string of from 1 to 15 characters.

You can use the **prcnam** argument to awaken only processes in the same UIC group as the calling process because process names are unique to UIC groups, and VMS uses the UIC group number of the calling process to interpret the process name specified by the **prcnam** argument. You must use the **pidadr** argument to awaken processes in other UIC groups.

If you specify neither the **pidadr** nor **prcnam** argument, the wakeup request is issued on behalf of the calling process.

daytim

VMS usage: **date_time**
type: **quadword (unsigned)**
access: **read only**
mechanism: **by reference**

Time at which the process is to be awakened. The **daytim** argument is the address of a quadword containing this time in the system 64-bit time format. A positive time value specifies an absolute time at which the specified process is to be awakened. A negative time value specifies an offset (delta time) from the current time.

reptim

VMS usage: **date_time**
type: **quadword (unsigned)**
access: **read only**
mechanism: **by reference**

Time interval at which the wakeup request is to be repeated. The **reptim** argument is the address of a quadword containing this time interval. The time interval must be expressed in delta time format.

The time interval specified cannot be less than 10 milliseconds; if it is, $SCHDWK automatically increases it to 10 milliseconds.

If you do not specify **reptim**, a default value of 0 is used, which specifies that the wakeup request is not to be repeated.

DESCRIPTION

Depending on the operation, the calling process may need a certain privilege to use $SCHDWK:

- GROUP privilege to schedule wakeup requests for a process in the same group unless it has the same UIC.
- WORLD privilege to schedule wakeup requests for any other process in the system.

$SCHDWK uses the following system resources:

- The AST limit (ASTLM) quota of the calling process to schedule a wakeup request.
- System dynamic memory to allocate a timer queue entry.

If you issue one or more scheduled wakeup requests for a process that is not hibernating, a subsequent hibernate request by the target process completes immediately; that is, the process does not hibernate. No count is maintained of outstanding wakeup requests.

You can cancel scheduled wakeup requests that have not yet been processed by using the Cancel Wakeup ($CANWAK) service.

If a specified absolute time value has already passed and no repeat time is specified, the timer expires at the next clock cycle (within 10 milliseconds).

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The expiration time, repeat time, process name string or string descriptor cannot be read by the caller, or the process identification cannot be written by the caller.
SS$_EXQUOTA	The process has exceeded its AST limit quota.
SS$_INSFMEM	The system dynamic memory is insufficient for allocating a timer queue entry.
SS$_IVLOGNAM	The process name string has a length of 0 or has more than 15 characters.
SS$_IVTIME	The specified delta repeat time is a positive value, or an absolute time plus delta repeat time is less than the current time.
SS$_NONEXPR	The specified process does not exist, or an invalid process identification was specified.
SS$_NOPRIV	The process does not have the privilege to schedule a wakeup request for the specified process.

$SETAST Set AST Enable

The Set AST Enable service enables or disables the delivery of ASTs for the access mode from which the service call was issued.

FORMAT

SYS$SETAST *enbflg*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENT

enbflg
VMS usage: **boolean**
type: **byte (unsigned)**
access: **read only**
mechanism: **by value**

Value specifying whether ASTs are to be enabled. The **enbflg** argument is a byte containing this value. The value 1 enables AST delivery for the calling access mode; the value 0 disables AST delivery.

DESCRIPTION

When an image is executing in user mode, ASTs are enabled for all higher access modes.

If ASTs are disabled for a more privileged access mode, VMS cannot deliver ASTs for less privileged access modes until ASTs are enabled once again for the more privileged access mode. Therefore, a process that has disabled ASTs for a more privileged access mode must re-enable ASTs for that mode before returning to a less privileged access mode.

CONDITION VALUES RETURNED

SS$_WASCLR	The service completed successfully. AST delivery was previously disabled for the calling access mode.
SS$_WASSET	The service completed successfully. AST delivery was previously enabled for the calling access mode.

$SETEF Set Event Flag

The Set Event Flag service sets an event flag in a local or common event flag cluster. The condition value returned by $SETEF indicates whether the specified flag was previously set or clear. After the event flag is set, processes waiting for the event flag to be set resume execution.

FORMAT

SYS$SETEF *efn*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENT

efn

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

Number of the event flag to be set. The **efn** argument is a longword containing this number; however, $SETEF uses only the low-order byte.

Two local event flag clusters are local to the process: cluster 0 and cluster 1. Cluster 0 contains event flag numbers 0 to 31, and cluster 1 contains event flag numbers 32 to 63.

There are two common event flag clusters: cluster 2 and cluster 3. Cluster 2 contains event flag numbers 64 to 95, and cluster 3 contains event flag numbers 96 to 127.

CONDITION VALUES RETURNED

SS$_WASCLR	The service completed successfully. The specified event flag was previously 0.
SS$_WASSET	The service completed successfully. The specified event flag was previously 1.
SS$_ILLEFC	You specified an illegal event flag number.
SS$_UNASEFC	The process is not associated with the cluster containing the specified event flag.

$SETEXV Set Exception Vector

The Set Exception Vector service (1) assigns a condition handler address to the primary, secondary, or last chance exception vectors or (2) removes a previously assigned handler address from any of these three vectors.

FORMAT

SYS$SETEXV *[vector],[addres],[acmode],[prvhnd]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

vector

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

Vector for which a condition handler is to be established or removed. The **vector** argument is a longword value. The value 0 (the default) specifies the primary vector; the value 1, the secondary vector; and the value 2, the last chance exception vector.

addres

VMS usage: **procedure**
type: **procedure entry mask**
access: **call without stack unwinding**
mechanism: **by reference**

Condition handler address to be established for the exception vector specified by **vector**. The **addres** argument is a longword value containing the address of the entry mask to the condition handler routine.

If you do not specify **addres** or specify it as 0, the condition handler address already established for the specified vector is removed; that is, the contents of the longword vector is set to 0.

acmode

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

Access mode for which the exception vector is to be modified. The **acmode** argument is a longword containing the access mode. The $PSLDEF macro defines symbols for the four access modes.

The most privileged access mode used is the access mode of the caller. Exception vectors for access modes more privileged than the caller's access mode cannot be modified.

prvhnd

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

Previous condition handler address contained by the specified exception vector. The **prvhnd** argument is the address of a longword into which $SETEF writes the handler address.

DESCRIPTION

A process cannot modify a vector associated with a more privileged access mode.

VMS provides two different methods for establishing condition handlers:

- Using the call stack associated with each access mode. Each call frame includes a longword to contain the address of a condition handler associated with that frame. The RTL routine LIB$ESTABLISH establishes a condition handler; the RTL routine LIB$REVERT removes a handler.
- Using the software exception vectors (by using $SETEXV) associated with each access mode. These vectors are set aside in the control region (P1 space) of the process.

The modular properties associated with the first method do not apply to the second. The software exception vectors are intended primarily for performance monitors and debuggers. For example, the primary exception vector and the last chance exception vector are used by the VMS Debugger for user-mode access, and DCL uses the last chance exception vector for supervisor-mode access.

User-mode exception vectors are canceled at image exit.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The longword to receive the previous contents of the vector cannot be written by the caller.

$SETIME Set System Time

The Set System Time service (1) changes the value of or (2) recalibrates the system time.

The system time is defined by a quadword value that specifies the number of 100 nanosecond intervals since 00:00 o'clock, November 17, 1858.

The system time is the reference used for nearly all timer-related software activities in VMS.

FORMAT

SYS$SETIME *[timadr]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENT

timadr
VMS usage: **date_time**
type: **quadword (unsigned)**
access: **read only**
mechanism: **by reference**

New absolute time value for the system time, specifying the number of 100 nanosecond intervals since 00:00 o'clock, November 17, 1858. The **timadr** argument is the address of a quadword containing the new system time value. A negative (delta) time value is invalid.

If you do not specify **timadr** or specify it as 0, $SETIME recalibrates the system time using the time-of-year clock.

DESCRIPTION

To set the system time, the calling process must have OPER and LOG_IO privileges.

After changing or recalibrating the system clock, $SETIME updates the timer queue by adjusting each element in the timer queue by the difference between the previous system time and the new system time.

The $SETIME service saves the new time (for future bootstrap operations) in the system image SYS$SYSTEM:SYS.EXE. To save the time, the service assigns a channel to the system boot device and calls the $QIOW service. You need the LOG_IO user privilege to perform this operation.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_IVTIME	The caller specified no time value or a negative time value and an invalid processor clock was found.
SS$_ACCVIO	The quadword that contains the new system time value cannot be read by the caller.
SS$_NOIOCHAN	No I/O channel is available for assignment.
SS$_NOPRIV	The process does not have the privileges to set the system time.

$SETIMR Set Timer

The Set Timer service sets the timer to expire at a specified time. When the timer expires, an event flag is set and (optionally) an AST routine executes.

FORMAT

SYS$SETIMR *[efn] ,daytim ,[astadr] ,[reqidt] ,[flags]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

efn

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

Event flag to be set when the timer expires. The **efn** argument is a longword value containing the number of the event flaG; however, $SETIMR uses only the low-order byte. If you do not specify **efn**, event flag 0 is set.

When $SETIMR first executes, it clears the specified event flag or event flag 0.

daytim

VMS usage: **date_time**
type: **quadword (unsigned)**
access: **read only**
mechanism: **by reference**

Time at which the timer expires. The **daytim** argument is the address of a quadword time value. A positive time value specifies an absolute time at which the timer expires; a negative time value specifies an offset (delta time) from the current time.

If a specified absolute time value has already passed, the timer expires at the next clock cycle, which is within 10 milliseconds.

The Convert ASCII String to Binary Time ($BINTIM) service converts an ASCII string time value to the quadword time value required by $SETIMR.

astadr

VMS usage: **ast_procedure**
type: **procedure entry mask**
access: **call without stack unwinding**
mechanism: **by reference**

AST service routine that is to execute when the timer expires. The **astadr** argument is the address of the entry mask of this routine. If you do not specify **astadr** or specify it as 0 (the default), no AST routine executes.

The AST routine, if specified, executes at the access mode of the caller.

reqidt

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

Identification of the timer request. The **reqidt** argument is a longword value containing a number that uniquely identifies the timer request. If you do not specify **reqidt**, the value 0 is used.

To cancel a timer request, the identification of the timer request (as specified by **reqidt** in $SETIMR) is passed to the Cancel Timer ($CANTIM) service (as the **reqidt** argument).

If you want to cancel specific timer requests, but not all timer requests, be sure to specify a nonzero value for **reqidt** in the $SETIMR call; $CANTIM interprets an identification value of zero as a request to cancel all timer requests.

You can specify unique values for **reqidt** for each timer request, or give the same value can be given to related timer requests. This allows for selective cancelling of a single timer request, a group of related timer requests, or all timer requests.

If you specify the **astadr** argument in the $SETIMR call, the value specified by the **reqidt** argument is passed as a parameter to the AST routine. If the AST routine requires more than one parameter, specify an address for the value of **reqidt**; the AST routine can then interpret that address as the beginning of a list of parameters.

flags

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

Longword of bit flags for the set timer operation. Currently, only bit 0 is used for the **flags** argument. When the low bit (bit 0) is set, it indicates that this timer request should be in units of CPU time, rather than elapsed time. When bit 0 is clear (the default), the timer request is in units of elapsed time.

DESCRIPTION

The Set Timer service requires dynamic memory and uses the process's timer queue entries (TQELM) quota. If you specify an AST routine, the service uses the AST limit (ASTLM) quota of the process.

The $SETIMR service executes at the access mode of the caller, as does the AST routine, if one is specified.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The expiration time cannot be read by the caller.
SS$_EXQUOTA	The process exceeded its quota for timer entries or its AST limit quota; or the system dynamic memory is insufficient for completing the request.
SS$_ILLEFC	You specified an illegal event flag number.
SS$_INSFMEM	The dynamic memory is insufficient for allocating a timer queue entry.
SS$_UNASEFC	The process is not associated with the cluster containing the specified event flag.

$SETPRA Set Power Recovery AST

The Set Power Recovery AST service establishes a routine to receive control after a power recovery is detected.

FORMAT

SYS$SETPRA *astadr ,[acmode]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

astadr

VMS usage: **ast_procedure**
type: **procedure entry mask**
access: **call without stack unwinding**
mechanism: **by reference**

Power recovery AST routine to receive control when a power recovery is detected. The **astadr** argument is the address of the entry mask of this routine.

If you specify **astadr** as 0, an AST is not delivered to the process when a power recovery is detected.

The system passes one parameter to the specified AST routine. This parameter is a longword value containing the length of time that the power was off, expressed as the number of 1/100th of a second intervals that have elapsed.

acmode

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

Access mode at which the power recovery AST routine is to execute. The **acmode** argument is a longword containing the access mode. The $PSLDEF macro defines symbols for the four access modes.

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

$SETPRA

DESCRIPTION

The $SETPRA system service uses the AST limit (ASTLM) quota of the process.

You can specify only one power recovery AST routine for a process. The AST entry point address is cleared at image exit.

The entry and exit conventions for the power recovery AST routine are the same as for all AST service routines. These conventions are described in the *Introduction to VMS System Services*.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_EXQUOTA	The process exceeded its quota for outstanding AST requests.

$SETPRI Set Priority

The Set Priority service changes the base priority of the process. The base priority is used to determine the order in which executable processes are to run.

FORMAT

SYS$SETPRI *[pidadr] ,[prcnam] ,pri ,[prvpri]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

pidadr

VMS usage: **process_id**
type: **longword (unsigned)**
access: **modify**
mechanism: **by reference**

Process identification (PID) of the process whose priority is to be set. The **pidadr** argument is the address of the PID.

prcnam

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

Process name of the process whose priority is to be changed. The **prcnam** argument is the address of a character string descriptor pointing to a 1- to 15-character process name string.

You can use the **prcnam** argument only on behalf of processes in the same UIC group as the calling process. To set the priority for processes in other groups, you must specify the **pidadr** argument.

If you specify neither the **pidadr** nor **prcnam** argument, $SETPRI sets the base priority of the calling process.

pri

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

New base priority to be established for the process. The **pri** argument is a longword value containing the new priority. Priorities that are not real time

are in the range 0 through 15; real-time priorities are in the range 16 through 31.

If the specified priority is higher than the base priority of the target process, and if the caller does not have ALTPRI privilege, then the base priority of the target process is used.

prvpri

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

Base priority of the process before the call to $SETPRI. The **prvpri** argument is the address of a longword into which $SETPRI writes the previous base priority of the process.

DESCRIPTION

Depending on the operation, the calling process may need one of the following privileges to use $SETPRI:

- GROUP privilege to change the priority of a process in the same group, unless the target process has the same UIC as the calling process.
- WORLD privilege to change the priority of any other process in the system.
- ALTPRI privilege to set any process's priority to a value greater than the target process's initial base priority.

The base priority of a process remains in effect until specifically changed or until the process is deleted.

If a process does not have ALTPRI privilege and attempts to set a priority higher than the base priority of the target process, the priority is set to the base priority of the target process, and the status code SS$_NORMAL is returned.

To determine the priority set by the $SETPRI service, use the Get Job/Process Information ($GETJPI) service.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The process name string or string descriptor cannot be read by the caller, or the process identification or previous priority longword cannot be written by the caller.
SS$_IVLOGNAM	The process name string has a length of 0 or has more than 15 characters.
SS$_NONEXPR	The specified process does not exist, or an invalid process identification was specified.
SS$_NOPRIV	The process does not have the privilege to affect other processes.

$SETPRN Set Process Name

The Set Process Name service allows a process to establish or to change its own process name.

FORMAT

SYS$SETPRN *[prcnam]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENT

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

Process name to be given to the calling process. The **prcnam** argument is the address of a character string descriptor pointing to a 1- to 15-character process name string. If you do not specify **prcnam**, the calling process is given no name.

DESCRIPTION

A process name remains in effect until you change it (using $SETPRN) or until the process is deleted.

Process names provide an identification mechanism for processes executing with the same group number. A process can also be identified by its process identification (PID).

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The process name string or string descriptor cannot be read by the caller.
SS$_DUPLNAM	The specified process name duplicates one already specified within that group.
SS$_IVLOGNAM	The specified process name has a length of 0 or has more than 15 characters.

$SETPRT Set Protection on Pages

The Set Protection on Pages service allows a process to change the protection on a page or range of pages.

FORMAT

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

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

inadr

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

Starting and ending virtual addresses of the range of pages whose protection is to be changed. The **inadr** argument is the address of a 2-longword array containing, in order, the starting and ending process virtual addresses. Only the virtual page number portion of each virtual address is used; the low-order 9 bits are ignored.

If the starting and ending virtual addresses are the same, the protection is changed for a single page.

retadr

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

Starting and ending virtual addresses of the range of pages whose protection was actually changed by $SETPRT. The **retadr** argument is the address of a 2-longword array containing, in order, the starting and ending process virtual addresses.

If an error occurs while the protection is being changed, $SETPRT writes into **retadr** the range of pages that were successfully changed before the error occurred. If no pages were affected before the error occurred, $SETPRT writes the value –1 into each longword of the 2-longword array.

acmode

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

Access mode associated with the call to $SETPRT. The **acmode** argument is a longword containing the access mode. The $PSLDEF macro defines symbols for the four access modes.

The $SETPRT service uses whichever of the following two access modes is least privileged: (1) the access mode specified by **acmode** or (2) the access mode of the caller. To change the protection of any page in the specified range, the resultant access mode must be equal to or more privileged than the access mode of the owner of that page.

prot

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

Page protection to be assigned to the specified pages. The **prot** argument is a longword value containing the protection code. Only bits 0 to 3 are used; bits 4 to 31 are ignored.

The $PRTDEF macro defines the following symbolic names for the protection codes.

Symbolic Name	Description
PRT$C_NA	No access
PRT$C_KR	Kernel read only
PRT$C_KW	Kernel write
PRT$C_ER	Executive read only
PRT$C_EW	Executive write
PRT$C_SR	Supervisor read only
PRT$C_SW	Supervisor write
PRT$C_UR	User read only
PRT$C_UW	User write
PRT$C_ERKW	Executive read; kernel write
PRT$C_SRKW	Supervisor read; kernel write
PRT$C_SREW	Supervisor read; executive write
PRT$C_URKW	User read; kernel write
PRT$C_UREW	User read; executive write
PRT$C_URSW	User read; supervisor write

If you specify the protection as 0, the protection defaults to kernel read only.

prvprt

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

Protection previously assigned to the last page in the range. The **prvprt** argument is the address of a byte into which $SETPRT writes the protection of this page. The **prvprt** argument is useful only when protection for a single page is being changed.

DESCRIPTION

If a process changes any pages in a private section from read only to read/write, $SETPRT uses the paging file (PGFLQUOTA) quota of the process.

For pages in global sections, the new protection can alter only copy-on-reference pages.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The input address array cannot be read by the caller; the output address array or the byte to receive the previous protection cannot be written by the caller; or an attempt was made to change the protection of a nonexistent page.
SS$_EXQUOTA	The process exceeded its paging file quota while changing a page in a read-only private section to a read/write page.
SS$_IVPROTECT	The specified protection code has a numeric value of 1 or is greater than 15.
SS$_LENVIO	A page in the specified range is beyond the end of the program or control region.
SS$_NOPRIV	A page in the specified range is in the system address space.
SS$_PAGOWNVIO	The process attempted to change the protection on a page owned by a more privileged access mode.

$SETPRV Set Privileges

The Set Privileges service enables or disables specified privileges for the calling process.

FORMAT

SYS$SETPRV *[enbflg],[prvadr],[prmflg],[prvprv]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

enbflg

VMS usage: **boolean**
type: **byte (unsigned)**
access: **read only**
mechanism: **by value**

Indicator specifying whether the specified privileges are to be enabled or disabled. The **enbflg** argument is a byte value. The value 1 indicates that the privileges specified in the **prvadr** argument are to be enabled. The value 0 (the default) indicates that the privileges are to be disabled.

prvadr

VMS usage: **mask_privileges**
type: **quadword (unsigned)**
access: **read only**
mechanism: **by reference**

Privileges to be enabled or disabled for the calling process. The **prvadr** argument is the address of a quadword bit vector wherein each bit corresponds to a privilege that is to be enabled or disabled.

Each bit has a symbolic name. The $PRVDEF macro defines these names. You form the bit vector by specifying the symbolic name of each desired privilege in a logical OR operation. Table SYS–6 provides the symbolic name and description of each privilege.

Table SYS–6 User Privileges

Privilege	Symbolic Name	Description
ALLSPOOL	PRV$M_ALLSPOOL	Allocate a spooled device
BUGCHK	PRV$M_BUGCHK	Make bugcheck error log entries
BYPASS	PRV$M_BYPASS	Bypass UIC-based protection
CMEXEC	PRV$M_CMEXEC	Change mode to executive
CMKRNL	PRV$M_CMKRNL	Change mode to kernel
DETACH	PRV$M_DETACH	Create detached processes
DIAGNOSE	PRV$M_DIAGNOSE	May diagnose devices
DOWNGRADE	PRV$M_DOWNGRADE	May downgrade classification
EXQUOTA	PRV$M_EXQUOTA	May exceed quotas
GROUP	PRV$M_GROUP	Group process control
GRPNAM	PRV$M_GRPNAM	Place name in group logical name table
GRPPRV	PRV$M_GRPPRV	Group access by means of system protection field
LOG_IO	PRV$M_LOG_IO	Perform logical I/O operations
MOUNT	PRV$M_MOUNT	Issue mount volume QIO
NETMBX	PRV$M_NETMBX	Create a network device
ACNT	PRV$M_NOACNT	Create processes for which no accounting is done
OPER	PRV$M_OPER	All operator privileges
PFNMAP	PRV$M_PFNMAP	Map to section by physical page frame number
PHY_IO	PRV$M_PHY_IO	Perform physical I/O operations
PRMCEB	PRV$M_PRMCEB	Create permanent common event flag clusters
PRMGBL	PRV$M_PRMGBL	Create permanent global sections
PRMMBX	PRV$M_PRMMBX	Create permanent mailboxes
PSWAPM	PRV$M_PSWAPM	Change process swap mode
READALL	PRV$M_READALL	Possess read access to everything
SECURITY	PRV$M_SECURITY	May perform security functions
ALTPRI	PRV$M_SETPRI	Set (alter) any process priority
SETPRV	PRV$M_SETPRV	Set any process privileges
SHARE	PRV$M_SHARE	May assign a channel to a nonshared device
SHMEM	PRV$M_SHMEM	Allocate structures in memory shared by multiple processors
SYSGBL	PRV$M_SYSGBL	Create system global sections
SYSLCK	PRV$M_SYSLCK	Queue systemwide locks
SYSNAM	PRV$M_SYSNAM	Place name in system logical name table

Table SYS–6 (Cont.) User Privileges

Privilege	Symbolic Name	Description
SYSPRV	PRV$M_SYSPRV	Access files and other resources as if you have a system UIC
TMPMBX	PRV$M_TMPMBX	Create temporary mailboxes
UPGRADE	PRV$M_UPGRADE	May upgrade classification
VOLPRO	PRV$M_VOLPRO	Override volume protection
WORLD	PRV$M_WORLD	World process control

Note that the names of the privilege bits PRV$M_NOACNT and PRV$M_SETPRI correspond to the names of the DCL privileges ACNT and ALTPRI, yet have different names.

If you do not specify **prvadr** or specify it as 0, the privileges are not altered.

get jobprmflg

VMS usage: **boolean**
type: **byte (unsigned)**
access: **read only**
mechanism: **by value**

Indicator specifying whether the privileges are to be affected permanently or temporarily. The **prmflg** argument is a byte value. The value 1 specifies that the privileges are to be affected permanently, that is, until you change them again by using $SETPRV or until the process is deleted. The value 0 (the default) specifies that the privileges are to be affected temporarily, that is, until the current image exits (at which time the permanently enabled privileges of the process will be restored).

prvprv

VMS usage: **mask_privileges**
type: **quadword (unsigned)**
access: **write only**
mechanism: **by reference**

Privileges previously possessed by the calling process. The **prvprv** argument is the address of a quadword bit vector wherein each bit corresponds to a privilege that was previously either enabled or disabled. If you do not specify **prvprv** or specify it as 0, the previous privilege mask is not returned.

DESCRIPTION

To set a privilege permanently, the calling process must be authorized to set the specified privilege, or the process must be executing in kernel or executive mode.

To set a privilege temporarily, one of the following three conditions must be true:

- The calling process must be authorized to set the specified privilege.
- The calling process must be executing in kernel or executive mode.
- The image currently executing must be one that was installed with the specified privilege.

VMS maintains four separate privilege masks for each process:

- AUTHPRIV—Privileges that the process is authorized to enable, as designated by the system manager or the process creator. The AUTHPRIV mask never changes during the life of the process.
- PROCPRIV—Privileges that are designated as permanently enabled for the process. The PROCPRIV mask can be modified by $SETPRV.
- IMAGPRIV—Privileges with which the current image is installed.
- CURPRIV—Privileges that are currently enabled. The CURPRIV mask can be modified by $SETPRV.

When a process is created, its AUTHPRIV, PROCPRIV, and CURPRIV masks have the same contents. Whenever a system service (other than $SETPRV) must check the process privileges, that service checks the CURPRIV mask.

When a process runs an installed image, the privileges with which that image was installed are enabled in the CURPRIV mask. When the installed image exits, the PROCPRIV mask is copied to the CURPRIV mask.

The $SETPRV service can set bits only in the CURPRIV and PROCPRIV mask, but $SETPRV checks the AUTHPRIV mask to see whether a process can set specified privilege bits in the CURPRIV or PROCPRIV masks. Consequently, a process can give itself the SETPRV privilege only if this privilege is enabled in the AUTHPRIV mask.

You can obtain each of a process's four privilege masks by calling the Get Job/Process Information ($GETJPI) service and specifying the desired privilege mask or masks as item codes in the **itmlst** argument. You construct the item code for a privilege mask by prefixing the name of the privilege mask with the characters **JPI$_** (for example, JPI$_CURPRIV is the item code for the current privilege mask).

The DCL command SET PROCESS/PRIVILEGES also enables or disables specified privileges; refer to the *VMS DCL Dictionary* for details.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully. All privileges were enabled or disabled as specified.
SS$_NOTALLPRIV	The service completed successfully. Not all specified privileges were enabled; see the Description section for details.
SS$_ACCVIO	The privilege mask cannot be read or the previous privilege mask cannot be written by the caller.

$SETRWM Set Resource Wait Mode

The Set Resource Wait Mode service allows a process to specify what action system services should take when system resources required for their execution are unavailable.

When resource wait mode is enabled, system services wait for the required system resources to become available and then continue execution.

When resource wait mode is disabled, system services return to the caller when required system resources are unavailable.

The condition value returned by $SETRWM indicates whether resource wait mode was previously enabled or previously disabled.

FORMAT

SYS$SETRWM *[watflg]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENT

watflg

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

Indicator specifying whether system services should wait for required resources. The **watflg** argument is a longword value. The value 0 (the default) specifies that system services should wait until resources needed for their execution become available. The value 1 specifies that system services should return failure status immediately when resources needed for their execution are unavailable.

VMS enables resource wait mode for all processes. You can disable resource wait mode only by calling $SETRWM.

If resource wait mode is disabled, it remains disabled until it is explicitly re-enabled or until the process is deleted.

DESCRIPTION

The following system resources and process quotas are affected by resource wait mode:

- System dynamic memory
- UNIBUS adapter map registers
- Direct I/O limit (DIOLM) quota
- Buffered I/O limit (BIOLM) quota
- Buffered I/O byte count limit (BYTLM) quota

CONDITION VALUES RETURNED

SS$_WASCLR	The service completed successfully. Resource wait mode was previously enabled.
SS$_WASSET	The service completed successfully. Resource wait mode was previously disabled.

$SETSFM Set System Service Failure Exception Mode

The Set System Service Failure Exception Mode service allows a process to specify whether VMS should generate a software exception when a system service returns an error or severe error condition value to the calling process.

The $SETSFM indicates in the condition value it returns whether system service exception mode was enabled or disabled prior to the call to $SETSFM.

Initially, system service failure exception mode is disabled, so the caller should explicitly test for successful completion following a system service call.

FORMAT

SYS$SETSFM *[enbflg]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENT

enbflg
VMS usage: **boolean**
type: **byte (unsigned)**
access: **read only**
mechanism: **by value**

Number specifying whether the system service failure exception mode is to be enabled. The **enbflg** argument is a byte value. The value 1 specifies that the system service failure exception mode is enabled. The value 0 (the default) specifies that the system service failure exception mode is disabled.

DESCRIPTION

When enabled, a software exception is generated when a system service returns an error or severe error condition value. System service failure exceptions are generated only if the service call originated from user mode. You can call the $SETSFM service, however, from any access mode.

If enabled, system service failure exception mode remains enabled until explicitly disabled or until the image exits. You can specify a condition handler in the first longword of the procedure call stack or with the Set Exception Vector ($SETEXV) service. If you do not specify a condition handler, a default system handler is used. This condition handler causes the image to exit and then displays the exit status.

The argument list provided to the condition handler contains the code SS$_SSFAIL in the condition name argument of the signal array.

For an explanation and examples of condition handling routines, the format of the argument lists passed to the condition handler, and a discussion of the appropriate actions a condition handler may take, see the *Introduction to VMS System Services*.

CONDITION VALUES RETURNED

SS$_WASCLR	The service completed successfully. Failure exceptions were previously disabled.
SS$_WASSET	The service completed successfully. Failure exceptions were previously enabled.

$SETSSF Set System Services Filter

The Set System Services Filter service inhibits user mode calls to certain system services. The $SETSSF service cannot inhibit the following services:

$ASCTIM	$BINTIM	$EXIT	$FAO
$FAOL	$PUTMSG	$UNWIND	

FORMAT

SYS$SETSSF *[mask]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENT

mask

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

Category of system services that are to be inhibited for user-mode calls. The **mask** argument is a longword value of which only the first byte is significant. The first byte is a bit vector wherein a bit, when set, specifies a category of system service to be inhibited. Only bits 0 and 1 are used; bits 2 to 7 are reserved.

When bit 0 is set, all system services, including user-written system services, are inhibited except those listed previously.

When bit 1 is set, all system services, including user-written system services, are inhibited except the following and those listed previously.

$ADJSTK	$CRETVA	$DELTVA	$GETMSG
$SETSFM			

Bit 1 inhibits fewer system services than bit 0. Specifically, bit 1 does not inhibit the system services required by condition-handling and image-rundown services, whereas bit 0 does.

DESCRIPTION

To call $SETSSF successfully, the access mode of the caller must be equal to or more privileged than supervisor-mode access, and the SYSGEN parameter SSINHIBIT must be set when the system is bootstrapped.

If a system service that has been inhibited is called from user mode, one of the following two condition values is returned:

SS$_INHCHME	You called a disabled executive mode system service.
SS$_INHCHMK	You called a disabled kernel mode system service.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_NOPRIV	The process does not have the privilege to call the service.

$SETSTK Set Stack Limits

The Set Stack Limits service allows a process to change the size of its supervisor, executive, and kernel stacks by altering the values in the stack limit and base arrays held in P1 (per-process) space.

FORMAT

SYS$SETSTK *inadr ,[retadr] ,[acmode]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

inadr

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

Range of addresses that express the stack's new limits. The **inadr** argument is the address of a 2-longword array containing, in order, the address of the top of the stack and the address of the base of the stack. Because stacks in P1 space expand from high to low addresses, the address of the base of the stack must be greater than the address of the top of the stack.

retadr

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

Range of addresses that express the stack's previous limits. The **retadr** argument is the address of a 2-longword array into which $SETSTK writes, in the first longword, the previous address of the top of the stack and, in the second longword, the previous address of the base of the stack.

acmode

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

Access mode of the stack to be altered. The **acmode** argument is a longword containing the access mode. The $PSLDEF macro defines symbols for the four access modes. The most privileged access mode used is the access mode of the caller.

If **acmode** specifies user mode, $SETSTK performs no operation and returns the SS$_NORMAL condition value.

DESCRIPTION

The calling process can adjust the size of stacks only for access modes that are equal to or less privileged than the access mode of the calling process.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The input address array cannot be read by the caller; the input range is invalid; or the return address array cannot be written by the caller.

$SETSWM Set Process Swap Mode

The Set Process Swap Mode service allows a process to control whether it can be swapped out of the balance set.

When process swap mode is enabled, the process can be swapped out; when disabled, the process remains in the balance set until (1) process swap mode is re-enabled or (2) the process is deleted.

The $SETSWM service returns a condition value indicating whether process swap mode was enabled or disabled prior to the call to $SETSWM.

To lock some but not necessarily all process pages into the balance set, use the Lock Pages in Memory ($LCKPAG) service.

FORMAT

SYS$SETSWM *[swpflg]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENT

swpflg

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

Indicator specifying whether the process can be swapped. The **swpflg** argument is a longword value. The value 0 (the default) enables process swap mode, meaning the process can be swapped. The value 1 disables process swap mode, meaning the process cannot be swapped.

DESCRIPTION

To change its process swap mode, the calling process must have PSWAPM privilege.

CONDITION VALUES RETURNED

SS$_WASCLR	The service completed successfully. The process was not previously locked in the balance set.
SS$_WASSET	The service completed successfully. The process was previously locked in the balance set.
SS$_NOPRIV	The process does not have the necessary PSWAPM privilege.

$SETUAI Set User Authorization Information

The Set User Authorization Information service is used to modify the user authorization file (UAF) record for a specified user.

FORMAT

SYS$SETUAI *[nullarg] ,[nullarg] ,usrnam ,itmlst ,[nullarg] ,[nullarg] ,[nullarg]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

nullarg

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

Place-holding argument reserved by DIGITAL.

usrnam

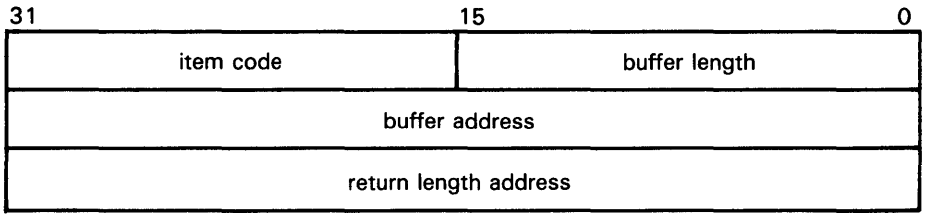
VMS usage: **char_string**
type: **character-coded text string**
access: **read only**
mechanism: **by descriptor—fixed-length string descriptor**

Name of the user whose user authorization file (UAF) record is modified. The **usrnam** argument is the address of a descriptor pointing to a character text string containing the user name. The user name string may contain a maximum of 12 alphanumeric characters.

itmlst

VMS usage: **item_list_3**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Item list specifying which information from the specified user's user authorization file (UAF) record is to be modified. The **itmlst** argument is the address of a list of one or more item descriptors, each of which specifies an item code. The item list is terminated by the item code 0 or by the longword 0. The following diagram depicts the structure of a single item descriptor.



ZK-1705-84

$SETUAI Item Descriptor Fields

buffer length

A word specifying the length (in bytes) of the buffer in which $SETUAI is to write the information. The length of the buffer varies depending on the item code specified in the **item code** field of the item descriptor and is given in the description of each item code. If the value of **buffer length** is too small, $SETUAI truncates the data.

item code

A word containing a user-supplied symbolic code specifying the item of information that $SETUAI is to set. The $UAIDEF macro defines these codes, which have the following format:

UAI$_code

Each item code is described under $SETUAI Item Codes.

buffer address

A longword address of the buffer that specifies the information to be set by $SETUAI.

return length address

A longword containing the user-supplied address of a word in which $SETUAI writes the length in bytes of the information it actually set.

$SETUAI Item Codes

UAI$_ACCOUNT

When you specify UAI$_ACCOUNT, $SETUAI sets, as a blank-filled character string, the account name of the user.

Because an account name can include up to 8 characters plus a size-byte prefix, the buffer length field of the item descriptor should specify 9 (bytes).

UAI$_ASTLM

When you specify UAI$_ASTLM, $SETUAI sets the AST queue limit.

Because this decimal number is a word in length, the buffer length field in the item descriptor should specify 2 (bytes).

UAI$_BATCH_ACCESS_P

When you specify UAI$_BATCH_ACCESS_P, $SETUAI sets, as a 3-byte value, the range of times during which batch access is permitted for primary days. Each bit set represents a 1-hour period, from bit 0 as midnight to 1 a.m., to bit 23 as 11 p.m. to midnight.

The buffer length field in the item descriptor should specify 3 (bytes).

UAI$_BATCH_ACCESS_S

When you specify UAI$_BATCH_ACCESS_S, $SETUAI sets, as a 3-byte value, the range of times during which batch access is permitted for secondary days. Each bit set represents a 1-hour period, from bit 0 as midnight to 1 a.m., to bit 23 as 11 p.m. to midnight.

The buffer length field in the item descriptor should specify 3 (bytes).

UAI$_BIOLM

When you specify UAI$_BIOLM, $SETUAI sets the buffered I/O count limit.

Because this decimal number is a word in length, the buffer length field in the item descriptor should specify 2 (bytes).

UAI$_BYTLM

When you specify UAI$_BYTLM, $SETUAI sets the buffered I/O byte limit.

Because the buffered I/O count limit is a longword decimal number, the buffer length field in the item descriptor should specify 4 (bytes).

UAI$_CLITABLES

When you specify UAI$_CLITABLES, $SETUAI sets, as a character string, the name of the user-defined CLI table for the account, if any.

Because the CLI table name can include up to 31 characters plus a size-byte prefix, the buffer length field of the item descriptor should specify 32 (bytes).

UAI$_CPUTIM

When you specify UAI$_CPUTIM, $SETUAI sets the maximum CPU time limit (per session) for the process in 10-millisecond units.

Because the maximum CPU time limit is a longword decimal number, the buffer length field in the item descriptor should specify 4 (bytes).

UAI$_DEFCLI

When you specify UAI$_DEFCLI, $SETUAI sets, as an RMS file name component, the name of the command language interpreter used to execute the specified batch job. The file specification set assumes the device name and directory SYS$SYSTEM and the file type EXE.

Because a file name can include up to 31 characters plus a size-byte prefix, the buffer length field in the item descriptor should specify 32 (bytes).

UAI$_DEFDEV

When you specify UAI$_DEFDEV, $SETUAI sets, as a 1- to 31-character string, the name of the default device.

Because the device name string can include up to 31 characters plus a size-byte prefix, the buffer length field in the item descriptor should specify 32 (bytes).

UAI$_DEFDIR

When you specify UAI$_DEFDIR, $SETUAI sets, as a 1- to 63-character string, the name of the default directory.

Because the directory name string can include up to 63 characters plus a size-byte prefix, the buffer length field in the item descriptor should specify 64 (bytes).

UAI$_DEF_PRIV

When you specify UAI$_DEF_PRIV, $SETUAI sets, as a quadword value, the default privileges for the user.

Because the default privileges are set as a quadword value, the buffer length field in the item descriptor should specify 8 (bytes).

UAI$_DFWSCNT

When you specify UAI$_DFWSCNT, $SETUAI sets the default working set size.

Because the default working set size is a longword decimal number, the buffer length field in the item descriptor should specify 4 (bytes).

UAI$_DIOLM

When you specify UAI$_DIOLM, $SETUAI sets the direct I/O count limit.

Because this decimal number is a word in length, the buffer length field in the item descriptor should specify 2 (bytes).

UAI$_DIALUP_ACCESS_P

When you specify UAI$_DIALUP_ACCESS_P, $SETUAI sets, as a 3-byte value, the range of times during which dialup access is permitted for primary days. Each bit set represents a 1-hour period, from bit 0 as midnight to 1 a.m., to bit 23 as 11 p.m. to midnight.

The buffer length field in the item descriptor should specify 3 (bytes).

UAI$_DIALUP_ACCESS_S

When you specify UAI$_DIALUP_ACCESS_S, $SETUAI sets, as a 3-byte value, the range of times during which dialup access is permitted for secondary days. Each bit set represents a 1-hour period, from bit 0 as midnight to 1 a.m., to bit 23 as 11 p.m. to midnight.

The buffer length field in the item descriptor should specify 3 (bytes).

UAI$_ENQLM

When you specify UAI$_ENQLM, $SETUAI sets the lock queue limit.

Because this decimal number is a word in length, the buffer length field in the item descriptor should specify 2 (bytes).

UAI$_EXPIRATION

When you specify UAI$_EXPIRATION, $SETUAI sets, as a quadword absolute time value, the expiration date and time of the account.

Because the absolute time value is a quadword in length, the buffer length field in the item descriptor should specify 8 (bytes).

UAI$_FILLM

When you specify UAI$_FILLM, $SETUAI sets the open file limit.

Because this decimal number is a word in length, the buffer length field in the item descriptor should specify 2 (bytes).

UAI$_FLAGS

When you specify UAI$_FLAGS, $SETUAI sets, as a longword bit vector, the various login flags set for the user.

Each flag is represented by a bit. The $UAIDEF macro defines the following symbolic names for these flags:

Symbolic Name	Description
UAI$V_AUDIT	All actions are audited.
UAI$V_AUTOLOGIN	User can only log in to terminals defined by the automatic login facility (ALF).
UAI$V_CAPTIVE	User is restricted to captive account.
UAI$V_DEFCLI	User is restricted to default command interpreter.
UAI$V_DISACNT	User account is disabled.
UAI$V_DISCTLY	User cannot use CTRL/Y.
UAI$V_DISMAIL	Announcement of new mail is suppressed.
UAI$V_DISRECONNECT	User cannot reconnect to existing processes.
UAI$V_DISREPORT	User will not receive last login messages.
UAI$V_DISWELCOME	User will not receive the login welcome message.
UAI$V_FORCE_EXP_ PWD_CHANGE	User is required to changed expired passwords.
UAI$V_GENPWD	User is required to use generated passwords.
UAI$V_LOCKPWD	SET PASSWORD command is disabled.
UAI$V_NOMAIL	Mail delivery to user is disabled.
UAI$V_PWD_EXPIRED	Primary password is expired.
UAI$V_PWD2_EXPIRED	Secondary password is expired.

UAI$_JTQUOTA

When you specify UAI$_JTQUOTA, $SETUAI sets the initial byte quota with which the jobwide logical name table is to be created.

Because this quota is a longword decimal number, the buffer length field in the item descriptor should specify 4 (bytes).

UAI$_LGICMD

When you specify UAI$_LGICMD, $SETUAI sets, as an RMS file specification, the name of the default login command file.

Because a file specification can include up to 63 characters plus a size-byte prefix, the buffer length field of the item descriptor should specify 64 (bytes).

UAI$_LOCAL_ACCESS_P

When you specify UAI$_LOCAL_ACCESS_P, $SETUAI sets, as a 3-byte value, the range of times during which local interactive access is permitted for primary days. Each bit set represents a 1-hour period, from bit 0 as midnight to 1 a.m., to bit 23 as 11 p.m. to midnight.

The buffer length field in the item descriptor should specify 3 (bytes).

UAI$_LOCAL_ACCESS_S

When you specify UAI$_LOCAL_ACCESS_S, $SETUAI sets, as a 3-byte value, the range of times during which local interactive access is permitted for secondary days. Each bit set represents a 1-hour period, from bit 0 as midnight to 1 a.m., to bit 23 as 11 p.m. to midnight.

The buffer length field in the item descriptor should specify 3 (bytes).

UAI$_MAXACCTJOBS

When you specify UAI$_MAXACCTJOBS, $SETUAI sets the maximum number of batch, interactive, and detached processes that can be active at one time for all users of the same account. The value 0 represents an unlimited number.

Because this decimal number is a word in length, the buffer length field in the item descriptor should specify 2 (bytes).

UAI$_MAXDETACH

When you specify UAI$_MAXDETACH, $SETUAI sets the detached process limit. The value 0 represents an unlimited number.

Because this decimal number is a word in length, the buffer length field in the item descriptor should specify 2 (bytes).

UAI$_MAXJOBS

When you specify UAI$_MAXJOBS, $SETUAI sets the active process limit. A value of 0 represents an unlimited number.

Because this decimal number is a word in length, the buffer length field in the item descriptor should specify 2 (bytes).

UAI$_NETWORK_ACCESS_P

When you specify UAI$_NETWORK_ACCESS_P, $SETUAI sets, as a 3-byte value, the range of times during which network access is permitted for primary days. Each bit set represents a 1-hour period, from bit 0 as midnight to 1 a.m., to bit 23 as 11 p.m. to midnight.

The buffer length field in the item descriptor should specify 3 (bytes).

UAI$_NETWORK_ACCESS_S

When you specify UAI$_NETWORK_ACCESS_S, $SETUAI sets, as a 3-byte value, the range of times during which network access is permitted for secondary days. Each bit set represents a 1-hour period, from bit 0 as midnight to 1 a.m., to bit 23 as 11 p.m. to midnight.

The buffer length field in the item descriptor should specify 3 (bytes).

UAI$_OWNER

When you specify UAI$_OWNER, $SETUAI sets, as a character string, the name of the owner of the account.

Because the owner name can include up to 31 characters plus a size-byte prefix, the buffer length field of the item descriptor should specify 32 (bytes).

UAI$_PBYTLM

When you specify UAI$_PBYTLM, $SETUAI sets the paged buffer I/O byte count limit.

Because the paged buffer I/O byte count limit is a longword decimal number, the buffer length field in the item descriptor should specify 4 (bytes).

UAI$_PGFLQUOTA

When you specify UAI$_PGFLQUOTA, $SETUAI sets the paging file quota.

Because the paging file quota is a longword decimal number, the buffer length field in the item descriptor should specify 4 (bytes).

UAI$_PRCCNT

When you specify UAI$_PRCCNT, $SETUAI sets the subprocess creation limit.

Because this decimal number is a word in length, the buffer length field in the item descriptor should specify 2 (bytes).

UAI$_PRI

When you specify UAI$_PRI, $SETUAI sets the default base priority.

Because this decimal number is a byte in length, the buffer length field in the item descriptor should specify 1 (byte).

UAI$_PRIMEDAYS

When you specify UAI$_PRIMEDAYS, $SETUAI sets, as a longword bit vector, the primary and secondary days of the week.

Each bit represents a day of the week, with the bit clear representing a primary day and the bit set representing a secondary day. The $UAIDEF macro defines the following symbolic names for these bits:

UAI$V_MONDAY
UAI$V_TUESDAY
UAI$V_WEDNESDAY
UAI$V_THURSDAY
UAI$V_FRIDAY
UAI$V_SATURDAY
UAI$V_SUNDAY

UAI$_PRIV

When you specify UAI$_PRIV, $SETUAI sets, as a quadword value, the names of the privileges the user holds.

Because the privileges are set as a quadword value, the buffer length field in the item descriptor should specify 8 (bytes).

UAI$_PWD

When you specify UAI$_PWD, $SETUAI sets, as a quadword value, the hashed primary password of the user.

Because the hashed primary password is set as a quadword value, the buffer length field in the item descriptor should specify 8 (bytes).

UAI$_PWD_LENGTH

When you specify UAI$_PWD_LENGTH, $SETUAI sets the minimum password length.

Because this decimal number is a byte in length, the buffer length field in the item descriptor should specify 1 (byte).

UAI$_PWD_LIFETIME

When you specify UAI$_PWD_LIFETIME, $SETUAI sets, as a quadword absolute time value, the password lifetime.

Because the absolute time value is a quadword in length, the buffer length field in the item descriptor should specify 8 (bytes).

UAI$_PWD2

When you specify UAI$_PWD2, $SETUAI sets, as a quadword value, the hashed secondary password of the user.

Because the hashed secondary password is set as a quadword value, the buffer length field in the item descriptor should specify 8 (bytes).

UAI$_QUEPRI

When you specify UAI$_QUEPRI, $SETUAI sets the maximum job queue priority in the range 0 through 31.

Because this decimal number is a byte in length, the buffer length field in the item descriptor should specify 1 (byte).

UAI$_REMOTE_ACCESS_P

When you specify UAI$_REMOTE_ACCESS_P, $SETUAI sets, as a 3-byte value, the range of times during which batch access is permitted for primary days. Each bit set represents a 1-hour period, from bit 0 as midnight to 1 a.m., to bit 23 as 11 p.m. to midnight.

The buffer length field in the item descriptor should specify 3 (bytes).

UAI$_REMOTE_ACCESS_S

When you specify UAI$_REMOTE_ACCESS_S, $SETUAI sets, as a 3-byte value, the range of times during which batch access is permitted for secondary days. Each bit set represents a 1-hour period, from bit 0 as midnight to 1 a.m., to bit 23 as 11 p.m. to midnight.

The buffer length field in the item descriptor should specify 3 (bytes).

UAI$_SHRFILLM

When you specify UAI$_SHRFILLM, $SETUAI sets the shared file limit.

Because this decimal number is a word in length, the buffer length field in the item descriptor should specify 2 (bytes).

UAI$_TQCNT

When you specify UAI$_TQCNT, $SETUAI sets the timer queue entry limit.

Because this decimal number is a word in length, the buffer length field in the item descriptor should specify 2 (bytes).

UAI$_UIC

When you specify UAI$_UIC, $SETUAI sets, as a longword, the user identification code (UIC), containing the following two word-length subfields.

Symbolic Name	Description
UIC$W_MEM	The member number subfield of the UIC
UIC$W_GRP	The group number subfield of the UIC

UAI$_USERNAME

When you specify UAI$_USERNAME, $SETUAI sets, as a blank-filled character string of up to 12 bytes, the user name of the owner of the specified job.

Because a user name can include up to 12 characters, the buffer length field of the item descriptor should specify 12 (bytes).

UAI$_WSEXTENT

When you specify UAI$_WSEXTENT, $SETUAI sets the working set extent specified for the job or queue.

Because the working set extent is a longword decimal number, the buffer length field in the item descriptor should specify 4 (bytes).

UAI$_WSQUOTA

When you specify UAI$_WSQUOTA, $SETUAI sets the working set quota for the specified user.

Because the working set quota is a longword decimal number, the buffer length field in the item descriptor should specify 4 (bytes).

DESCRIPTION

The following list determines the privileges you need to use the $SETUAI service:

- BYPASS or SYSPRV—allows modification of any record in the UAF (user authorization file).
- GRPPRV—allows modification of any record in the UAF whose UIC group matches that of the requester. A group manager with GRPPRV privilege is limited in the extent to which he may modify the UAF records of users in the same group; values such as privileges and quotas may only be changed if the modification does not exceed the values set in the group manager's UAF record.
- No privilege—does not allow access to any UAF record.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The item list or input buffer cannot be read by the caller; or the return length buffer, output buffer, or status block cannot be written by the caller.
SS$_BADPARAM	The function code is invalid; the item list contains an invalid item code; a buffer descriptor has an invalid length; or the reserved parameter has a nonzero value.
SS$_NOPRIV	The user does not have the privileges required to examine the authorization information for the specified user.

This service may also return RMS status codes associated with operations on indexed files. For a description of RMS status codes that are returned by this service, refer to the *VMS Record Management Services Manual.*

$SNDERR Send Message to Error Logger

The Send Message to Error Logger service writes a user-specified message to the system error log file, preceding it with the date and time.

FORMAT

SYS$SNDERR *msgbuf*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENT

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

Message to be written to the error log file. The **msgbuf** argument is the address of a character string descriptor pointing to the message text.

DESCRIPTION

To send a message to the error log file, the calling process must have BUGCHK privilege.

The $SNDERR service requires system dynamic memory.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The message buffer or buffer descriptor cannot be read by the caller.
SS$_INSFMEM	The system dynamic memory is insufficient for completing the service.
SS$_NOPRIV	The process does not have the required BUGCHK privilege.

$SNDJBC Send to Job Controller

The Send to Job Controller service creates, stops, and manages queues and the batch and print jobs in those queues. For a discussion of the types of queue supported by the VMS batch/print facility, see the DESCRIPTION section. The $SNDJBC and $GETQUI services together provide the user interface to the VMS Job Controller, which is the VMS queue and accounting manager.

The $SNDJBC service completes asynchronously; that is, it returns to the caller after queuing the request, without waiting for the operation to complete.

To synchronize the completion of most operations, you use the Send to Job Controller and Wait ($SNDJBCW) service. The $SNDJBCW service is identical to $SNDJBC in every way except that $SNDJBCW returns to the caller after the operation completes.

For a discussion of the types of queue supported by the VMS batch/print facility, see the DESCRIPTION section.

For additional information about system service completion, refer to the Synchronize ($SYNCH) service and to the *Introduction to VMS System Services*.

The $SNDJBC and $SNDJBCW services supersede the Send Message to Symbiont Manager ($SNDSMB) and Send Message to Accounting Manager ($SNDACC) services. You should write new programs using $SNDJBC or $SNDJBCW, instead of $SNDSMB or $SNDACC. You should convert old programs containing $SNDSMB or $SNDACC to use $SNDJBC or $SNDJBCW.

FORMAT

SYS$SNDJBC *[efn] ,func [,nullarg] [,itmlst] [,iosb] [,astadr] [,astprm]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

efn

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

Number of the event flag to be set when $SNDJBC completes. The **efn** argument is a longword containing this number; however, $SNDJBC uses only the low-order byte.

When you queue the request, $SNDJBC clears the specified event flag (or event flag 0 if **efn** was not specified). Then, when the operation completes, $SNDJBC sets the specified event flag (or event flag 0).

func

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

Function code specifying the function that $SNDJBC is to perform. The **func** argument is a word containing this function code. The $SJCDEF macro defines the names of each function code.

You may specify only one function code in a single call to $SNDJBC. Most function codes require or allow for additional information to be passed in the call. You pass this information by using the **itmlst** argument, which specifies a list of one or more item descriptors. Each item descriptor in turn specifies an item code, which modifies, restricts, or otherwise affects the action designated by the function code.

The following lists and describes each function code, and lists which item code or codes you must and may specify for each function code; descriptions of the item codes appear in the description of the **itmlst** argument.

$SNDJBC Function Codes with Their Valid Item Codes

SJC$_ABORT_JOB

This request aborts the execution of the current job from an output execution queue or the job you specified from a batch queue. By default, the job is deleted. However, for a restartable job, you can requeue it to the same queue or to another queue.

You must specify the following input item code:

SJC$_QUEUE

You must specify the following input item code for batch jobs:

SJC$_ENTRY_NUMBER

You may specify the following optional input or Boolean item codes:

SJC$_DESTINATION_QUEUE
SJC$_HOLD
SJC$_NO_HOLD
SJC$_PRIORITY
SJC$_REQUEUE

SJC$_ADD_FILE

This request adds a file to the open job owned by the requesting process. You use this operation as part of a sequence of calls to the $SNDJBC service to create a job with one or more files. The first call in the sequence specifies the SJC$_CREATE_JOB operation to create an open job. Each subsequent SJC$_ADD_FILE request associates an additional file with the job. Finally, you make a SJC$_CLOSE_JOB request to complete the batch or print job specification. To create a job that contains only one file, you can make a single call to $SNDJBC that specifies the SJC$_ENTER_FILE function code.

You must specify one of the following input item codes:

SJC$_FILE_IDENTIFICATION
SJC$_FILE_SPECIFICATION

You may specify the following input or Boolean item codes:

SJC$_DELETE_FILE	SJC$_NO_DELETE_FILE
SJC$_DOUBLE_SPACE	SJC$_NO_DOUBLE_SPACE
SJC$_FILE_BURST	SJC$_NO_FILE_BURST
SJC$_FILE_COPIES	
SJC$_FILE_FLAG	SJC$_NO_FILE_FLAG
SJC$_FILE_SETUP_MODULES	SJC$_NO_FILE_SETUP_MODULES
SJC$_FILE_TRAILER	SJC$_NO_FILE_TRAILER
SJC$_FIRST_PAGE	SJC$_NO_FIRST_PAGE
SJC$_LAST_PAGE	SJC$_NO_LAST_PAGE
SJC$_PAGE_HEADER	SJC$_NO_PAGE_HEADER
SJC$_PAGINATE	SJC$_NO_PAGINATE
SJC$_PASSALL	SJC$_NO_PASSALL

SJC$_ALTER_JOB

This request alters the parameters of an existing job that is not currently executing.

You must specify the following input item code:

SJC$_ENTRY_NUMBER

You may specify the following input or Boolean item codes:

SJC$_AFTER_TIME	SJC$_NO_AFTER_TIME
SJC$_CHARACTERISTIC_NAME	SJC$_NO_CHARACTERISTICS
SJC$_CHARACTERISTIC_NUMBER	
	SJC$_NO_CHECKPOINT_DATA
SJC$_CLI	SJC$_NO_CLI
SJC$_CPU_LIMIT	SJC$_NO_CPU_LIMIT
	SJC$_NO_DELETE_FILE
SJC$_DESTINATION_QUEUE	
SJC$_DOUBLE_SPACE	SJC$_NO_DOUBLE_SPACE
SJC$_FILE_BURST	SJC$_NO_FILE_BURST
SJC$_FILE_COPIES	
SJC$_FILE_FLAG	SJC$_NO_FILE_FLAG
SJC$_FILE_SETUP_MODULES	SJC$_NO_FILE_SETUP_MODULES
SJC$_FILE_TRAILER	SJC$_NO_FILE_TRAILER
SJC$_FIRST_PAGE	SJC$_NO_FIRST_PAGE
SJC$_FORM_NAME	
SJC$_FORM_NUMBER	

SJC$_HOLD	SJC$_NO_HOLD
SJC$_JOB_COPIES	
SJC$_JOB_NAME	
SJC$_LAST_PAGE	SJC$_NO_LAST_PAGE
SJC$_LOG_DELETE	SJC$_NO_LOG_DELETE
SJC$_LOG_QUEUE	
SJC$_LOG_SPECIFICATION	SJC$_NO_LOG_SPECIFICATION
SJC$_LOG_SPOOL	SJC$_NO_LOG_SPOOL
SJC$_LOWERCASE	SJC$_NO_LOWERCASE
SJC$_NOTE	SJC$_NO_NOTE
SJC$_NOTIFY	SJC$_NO_NOTIFY
SJC$_OPERATOR_REQUEST	SJC$_NO_OPERATOR_REQUEST
SJC$_PAGE_HEADER	SJC$_NO_PAGE_HEADER
SJC$_PAGINATE	SJC$_NO_PAGINATE
SJC$_PARAMETER_1 through 8	SJC$_NO_PARAMETERS
SJC$_PASSALL	SJC$_NO_PASSALL
SJC$_PRIORITY	
SJC$_QUEUE	
SJC$_RESTART	SJC$_NO_RESTART
SJC$_WSDEFAULT	SJC$_NO_WSDEFAULT
SJC$_WSEXTENT	SJC$_NO_WSEXTENT
SJC$_WSQUOTA	SJC$_NO_WSQUOTA

If you specify the SJC$_QUEUE item code, the $SNDJBC service verifies that the selected job entry exists on the specified queue before modifying the job.

SJC$_ALTER_QUEUE

This request alters the parameters of a queue. The execution of current jobs is unaffected.

You must specify the following input item code:

SJC$_QUEUE

You may specify the following input or Boolean item codes:

SJC$_BASE_PRIORITY	
SJC$_CHARACTERISTIC_NAME	SJC$_NO_CHARACTERISTICS
SJC$_CHARACTERISTIC_NUMBER	
SJC$_CLOSE_QUEUE	
SJC$_CPU_DEFAULT	SJC$_NO_CPU_DEFAULT
SJC$_CPU_LIMIT	SJC$_NO_CPU_LIMIT
SJC$_DEFAULT_FORM_NAME	
SJC$_DEFAULT_FORM_NUMBER	
SJC$_FILE_BURST	SJC$_NO_FILE_BURST

SJC$_FILE_BURST_ONE	
SJC$_FILE_FLAG	SJC$_NO_FILE_FLAG
SJC$_FILE_FLAG_ONE	
SJC$_FILE_TRAILER	SJC$_NO_FILE_TRAILER
SJC$_FILE_TRAILER_ONE	
SJC$_FORM_NAME	
SJC$_FORM_NUMBER	
SJC$_GENERIC_SELECTION	SJC$_NO_GENERIC_SELECTION
SJC$_JOB_BURST	SJC$_NO_JOB_BURST
SJC$_JOB_FLAG	SJC$_NO_JOB_FLAG
SJC$_JOB_LIMIT	
SJC$_JOB_RESET_MODULES	SJC$_NO_JOB_RESET_MODULES
SJC$_JOB_SIZE_MAXIMUM	SJC$_NO_JOB_SIZE_MAXIMUM
SJC$_JOB_SIZE_MINIMUM	SJC$_NO_JOB_SIZE_MINIMUM
SJC$_JOB_SIZE_SCHEDULING	SJC$_NO_JOB_SIZE_SCHEDULING
SJC$_JOB_TRAILER	SJC$_NO_JOB_TRAILER
SJC$_OPEN_QUEUE	
SJC$_OWNER_UIC	
SJC$_PAGINATE	SJC$_NO_PAGINATE
SJC$_PROTECTION	
SJC$_QUEUE_DESCRIPTION	SJC$_NO_QUEUE_DESCRIPTION
SJC$_RECORD_BLOCKING	SJC$_NO_RECORD_BLOCKING
SJC$_RETAIN_ALL_JOBS	SJC$_NO_RETAIN_JOBS
SJC$_RETAIN_ERROR_JOBS	
SJC$_SWAP	SJC$_NO_SWAP
SJC$_WSDEFAULT	SJC$_NO_WSDEFAULT
SJC$_WSEXTENT	SJC$_NO_WSEXTENT
SJC$_WSQUOTA	SJC$_NO_WSQUOTA

SJC$_ASSIGN_QUEUE

This request assigns a logical queue to an execution queue. The SJC$_QUEUE item code specifies the logical queue; the SJC$_DESTINATION_QUEUE item code specifies the execution queue.

You must specify the following input item codes:

SJC$_QUEUE
SJC$_DESTINATION_QUEUE

SJC$_BATCH_CHECKPOINT

This request establishes a checkpoint in a batch job. No operation is performed if the requesting process is not a batch process.

You must specify the following input item code:

SJC$_CHECKPOINT_DATA

SJC$_CLOSE_DELETE

This request deletes the open job owned by the requesting process. No item codes are allowed.

SJC$_CLOSE_JOB

This request completes the specification of the open job owned by the requesting process and places the job in the queue specified in the SJC$_CREATE_JOB request that opened the job. If the SJC$_CLOSE_JOB request completes successfully, the job is no longer an open job; it becomes a normal batch or print job.

You may specify the following output item code:

SJC$_JOB_STATUS_OUTPUT

SJC$_CREATE_JOB

This request creates an open job for the requesting process. If the process already owns an open job, that job is deleted.

An open job is a batch or print job that has not yet been completely specified. After you make the SJC$_CREATE_JOB request to open the job, you can make subsequent calls to $SNDJBC using the SJC$_ADD_FILE function code to specify the files associated with the job. Finally, you can complete the job specification with an SJC$_CLOSE_JOB request. If the SJC$_CREATE_JOB operation completes successfully, the open job created is given an entry number; the job is not assigned to the queue specified in the SJC$_CREATE_JOB operation until the SJC$_CLOSE_JOB completes successfully.

You must specify the following input item code:

SJC$_QUEUE

You may specify the following input or Boolean item codes:

SJC$_ACCOUNT_NAME	
SJC$_AFTER_TIME	SJC$_NO_AFTER_TIME
SJC$_CHARACTERISTIC_NAME	SJC$_NO_CHARACTERISTICS
SJC$_CHARACTERISTIC_NUMBER	
SJC$_CLI	SJC$_NO_CLI
SJC$_CPU_LIMIT	SJC$_NO_CPU_LIMIT
SJC$_FILE_BURST	SJC$_NO_FILE_BURST
SJC$_FILE_BURST_ONE	
SJC$_FILE_FLAG	SJC$_NO_FILE_FLAG
SJC$_FILE_FLAG_ONE	
SJC$_FILE_TRAILER	SJC$_NO_FILE_TRAILER
SJC$_FILE_TRAILER_ONE	
SJC$_FORM_NAME	
SJC$_FORM_NUMBER	
SJC$_HOLD	SJC$_NO_HOLD
SJC$_JOB_COPIES	

SJC$_JOB_NAME	
SJC$_LOG_DELETE	SJC$_NO_LOG_DELETE
SJC$_LOG_QUEUE	
SJC$_LOG_SPECIFICATION	SJC$_NO_LOG_SPECIFICATION
SJC$_LOG_SPOOL	SJC$_NO_LOG_SPOOL
SJC$_LOWERCASE	SJC$_NO_LOWERCASE
SJC$_NOTE	SJC$_NO_NOTE
SJC$_NOTIFY	SJC$_NO_NOTIFY
SJC$_OPERATOR_REQUEST	SJC$_NO_OPERATOR_REQUEST
SJC$_PARAMETER_1 through 8	SJC$_NO_PARAMETERS
SJC$_PRIORITY	
SJC$_RESTART	SJC$_NO_RESTART
SJC$_UIC	
SJC$_USERNAME	
SJC$_WSDEFAULT	SJC$_NO_WSDEFAULT
SJC$_WSEXTENT	SJC$_NO_WSEXTENT
SJC$_WSQUOTA	SJC$_NO_WSQUOTA

You may specify the following output item code:

SJC$_ENTRY_NUMBER_OUTPUT

SJC$_CREATE_QUEUE

This request creates a queue. If the queue already exists and is not stopped, this request performs no operation. However, if the queue already exists and is stopped, the request alters the parameters of the queue based on the item codes specified in the request; if you specify the SJC$_CREATE_START item code, the request starts the queue.

You must specify the following input item code:

SJC$_QUEUE

You may specify the following input or Boolean item codes:

SJC$_BASE_PRIORITY	
SJC$_BATCH	SJC$_NO_BATCH
SJC$_CHARACTERISTIC_NAME	SJC$_NO_CHARACTERISTICS
SJC$_CHARACTERISTIC_NUMBER	
SJC$_CLOSE_QUEUE	
SJC$_CPU_DEFAULT	SJC$_NO_CPU_DEFAULT
SJC$_CPU_LIMIT	SJC$_NO_CPU_LIMIT
SJC$_CREATE_START	
SJC$_DEFAULT_FORM_NAME	
SJC$_DEFAULT_FORM_NUMBER	
SJC$_DEVICE_NAME	

SJC$_FILE_BURST	SJC$_NO_FILE_BURST
SJC$_FILE_BURST_ONE	
SJC$_FILE_FLAG	SJC$_NO_FILE_FLAG
SJC$_FILE_FLAG_ONE	
SJC$_FILE_TRAILER	SJC$_NO_FILE_TRAILER
SJC$_FILE_TRAILER_ONE	
SJC$_FORM_NAME	
SJC$_FORM_NUMBER	
SJC$_GENERIC_QUEUE	SJC$_NO_GENERIC_QUEUE
SJC$_GENERIC_SELECTION	SJC$_NO_GENERIC_SELECTION
SJC$_GENERIC_TARGET	
SJC$_JOB_BURST	SJC$_NO_JOB_BURST
SJC$_JOB_FLAG	SJC$_NO_JOB_FLAG
SJC$_JOB_LIMIT	
SJC$_JOB_RESET_MODULES	SJC$_NO_JOB_RESET_MODULES
SJC$_JOB_SIZE_MAXIMUM	SJC$_NO_JOB_SIZE_MAXIMUM
SJC$_JOB_SIZE_MINIMUM	SJC$_NO_JOB_SIZE_MINIMUM
SJC$_JOB_SIZE_SCHEDULING	SJC$_NO_JOB_SIZE_SCHEDULING
SJC$_JOB_TRAILER	SJC$_NO_JOB_TRAILER
SJC$_LIBRARY_SPECIFICATION	SJC$_NO_LIBRARY_SPECIFICATION
SJC$_OPEN_QUEUE	
SJC$_OWNER_UIC	
SJC$_PAGINATE	SJC$_NO_PAGINATE
SJC$_PRINTER	
SJC$_PROCESSOR	SJC$_NO_PROCESSOR
SJC$_PROTECTION	
SJC$_QUEUE_DESCRIPTION	SJC$_NO_QUEUE_DESCRIPTION
SJC$_RECORD_BLOCKING	SJC$_NO_RECORD_BLOCKING
SJC$_RETAIN_ALL_JOBS	SJC$_NO_RETAIN_JOBS
SJC$_RETAIN_ERROR_JOBS	
SJC$_SCSNODE_NAME	
SJC$_SERVER	
SJC$_SWAP	SJC$_NO_SWAP
SJC$_TERMINAL	SJC$_NO_TERMINAL
SJC$_WSDEFAULT	SJC$_NO_WSDEFAULT
SJC$_WSEXTENT	SJC$_NO_WSEXTENT
SJC$_WSQUOTA	SJC$_NO_WSQUOTA

SJC$_DEASSIGN_QUEUE

This request deassigns a logical queue from an execution queue.

You must specify the following input item code:

SJC$_QUEUE

SJC$_DEFINE_CHARACTERISTIC

This request defines a characteristic name and number and inserts this definition in the queue file. The characteristic name can be up to 31 characters in length. Each characteristic name must have a unique number in the range 0 to 127. If the characteristic name is already defined, the request alters the definition of the characteristic.

A job cannot execute on an execution queue unless the queue possesses all the characteristics possessed by the job; the queue may possess additional characteristics and the job will still execute.

You must specify the following input item codes:

SJC$_CHARACTERISTIC_NAME
SJC$_CHARACTERISTIC_NUMBER

SJC$_DEFINE_FORM

This request defines a form name and number, as well as other physical attributes of the paper stock used in printers, and inserts this definition into the system job queue file. If the form name is already defined, this request alters the definition of the form.

Forms are used only by output execution queues and print jobs. A print job cannot execute unless the stock name of a form specified for the queue is the same as the stock name specified for the job. The stock name of a form, which you specify by using the SJC$_FORM_STOCK item code, specifies the paper stock used by the printer. Other item codes specify printing parameters for a job such as the margins, length of paper, and so on.

Each form must have a unique number. Numbers can range from 0 to 999. When a new queue file is created, the system supplies the definition of a form named DEFAULT with number 0 and default characteristics.

You must specify the following input item codes:

SJC$_FORM_NAME
SJC$_FORM_NUMBER

You may specify the following input or Boolean item codes:

SJC$_FORM_DESCRIPTION	
SJC$_FORM_LENGTH	
SJC$_FORM_MARGIN_BOTTOM	
SJC$_FORM_MARGIN_LEFT	
SJC$_FORM_MARGIN_RIGHT	
SJC$_FORM_MARGIN_TOP	
SJC$_FORM_SETUP_MODULES	SJC$_NO_FORM_SETUP_MODULES
SJC$_FORM_SHEET_FEED	SJC$_NO_FORM_SHEET_FEED
SJC$_FORM_STOCK	
SJC$_FORM_TRUNCATE	SJC$_NO_FORM_TRUNCATE
SJC$_FORM_WIDTH	

SJC$_FORM_WRAP
SJC$_NO_FORM_WRAP
SJC$_PAGE_SETUP_MODULES
SJC$_NO_PAGE_SETUP_MODULES

SJC$_DELETE_CHARACTERISTIC

This request deletes the definition of a characteristic name.

You must specify the following input item code:

SJC$_CHARACTERISTIC_NAME

SJC$_DELETE_FORM

This request deletes the definition of a form name. There must be no queues or jobs that reference the form.

You must specify the following input item code:

SJC$_FORM_NAME

SJC$_DELETE_JOB

This request deletes a job from the system job queue file. If the job is currently executing, it is aborted.

You must specify the following input item code:

SJC$_ENTRY_NUMBER

You may specify the following input item code:

SJC$_QUEUE

If you specify the SJC$_QUEUE item code, the $SNDJBC service verifies that the selected job entry exists on the specified queue before deleting the job.

SJC$_DELETE_QUEUE

This request deletes a queue and all of the jobs in the queue. The queue must be stopped, and there must be no other queues or jobs that reference the queue.

You must specify the following input item code:

SJC$_QUEUE

SJC$_ENTER_FILE

This request creates a job containing one file and places the job in the specified queue. To create a job with more than one file, you must make a sequence of calls to the $SNDJBC service using the SJC$_CREATE_JOB, SJC$_ADD_FILE, and SJC$_CLOSE_JOB function codes.

You must specify the following input item code:

SJC$_QUEUE

You must specify one of the following input item codes:

SJC$_FILE_IDENTIFICATION
SJC$_FILE_SPECIFICATION

You may specify the following input or Boolean item codes:

SJC$_ACCOUNT_NAME	
SJC$_AFTER_TIME	SJC$_NO_AFTER_TIME
SJC$_CHARACTERISTIC_NAME	SJC$_NO_CHARACTERISTICS
SJC$_CHARACTERISTIC_NUMBER	
SJC$_CLI	SJC$_NO_CLI
SJC$_CPU_LIMIT	SJC$_NO_CPU_LIMIT
SJC$_DELETE_FILE	SJC$_NO_DELETE_FILE
SJC$_DOUBLE_SPACE	SJC$_NO_DOUBLE_SPACE
SJC$_FILE_BURST	SJC$_NO_FILE_BURST
SJC$_FILE_COPIES	
SJC$_FILE_FLAG	SJC$_NO_FILE_FLAG
SJC$_FILE_SETUP_MODULES	SJC$_NO_FILE_SETUP_MODULES
SJC$_FILE_TRAILER	SJC$_NO_FILE_TRAILER
SJC$_FIRST_PAGE	SJC$_NO_FIRST_PAGE
SJC$_FORM_NAME	
SJC$_FORM_NUMBER	
SJC$_HOLD	SJC$_NO_HOLD
SJC$_JOB_COPIES	
SJC$_JOB_NAME	
SJC$_LAST_PAGE	SJC$_NO_LAST_PAGE
SJC$_LOG_DELETE	SJC$_NO_LOG_DELETE
SJC$_LOG_QUEUE	
SJC$_LOG_SPECIFICATION	SJC$_NO_LOG_SPECIFICATION
SJC$_LOG_SPOOL	SJC$_NO_LOG_SPOOL
SJC$_LOWERCASE	SJC$_NO_LOWERCASE
SJC$_NOTE	SJC$_NO_NOTE
SJC$_NOTIFY	SJC$_NO_NOTIFY
SJC$_OPERATOR_REQUEST	SJC$_NO_OPERATOR_REQUEST
SJC$_PAGE_HEADER	SJC$_NO_PAGE_HEADER
SJC$_PAGINATE	SJC$_NO_PAGINATE
SJC$_PARAMETER_1 through 8	SJC$_NO_PARAMETERS
SJC$_PASSALL	SJC$_NO_PASSALL
SJC$_PRIORITY	
SJC$_RESTART	SJC$_NO_RESTART
SJC$_UIC	
SJC$_USERNAME	
SJC$_WSDEFAULT	SJC$_NO_WSDEFAULT
SJC$_WSEXTENT	SJC$_NO_WSEXTENT
SJC$_WSQUOTA	SJC$_NO_WSQUOTA

You may specify the following output item codes:

SJC$_ENTRY_NUMBER_OUTPUT
SJC$_JOB_STATUS_OUTPUT

SJC$_MERGE_QUEUE

This request requeues all jobs in the queue specified by the item code SJC$_QUEUE to the queue specified by the item code SJC$_DESTINATION_QUEUE. The execution of current jobs is unaffected.

You must specify the following input item codes:

SJC$_QUEUE
SJC$_DESTINATION_QUEUE

SJC$_PAUSE_QUEUE

This request pauses the execution of current jobs in the specified queue and prevents the starting of jobs in that queue.

You must specify the following input item code:

SJC$_QUEUE

SJC$_RESET_QUEUE

This request resets the specified queue by (1) terminating and deleting each executing job that is not restartable, (2) terminating and requeuing each executing job that is restartable, and (3) stopping the queue.

You must specify the following input item code:

SJC$_QUEUE

SJC$_START_ACCOUNTING

This request performs two functions. If you specify the SJC$_ACCOUNTING_TYPES item code, the request enables recording of the specified types of accounting records; if you do not specify SJC$_ACCOUNTING_TYPES, the request starts the accounting manager and opens the system accounting file.

You may specify the following input or Boolean item codes:

SJC$_ACCOUNTING_TYPES
SJC$_NEW_VERSION

SJC$_START_QUEUE

This request permits the starting of jobs in the specified queue. If the queue was paused, current jobs are resumed.

You must specify the following input item code:

SJC$_QUEUE

You may specify the following input or Boolean item codes:

SJC$_ALIGNMENT_MASK
SJC$_ALIGNMENT_PAGES
SJC$_BASE_PRIORITY
SJC$_BATCH SJC$_NO_BATCH

SJC$_CHARACTERISTIC_NAME	SJC$_NO_CHARACTERISTICS
SJC$_CHARACTERISTIC_NUMBER	
SJC$_CLOSE_QUEUE	
SJC$_CPU_DEFAULT	SJC$_NO_CPU_DEFAULT
SJC$_CPU_LIMIT	SJC$_NO_CPU_LIMIT
SJC$_DEFAULT_FORM_NAME	
SJC$_DEFAULT_FORM_NUMBER	
SJC$_DEVICE_NAME	
SJC$_FILE_BURST	SJC$_NO_FILE_BURST
SJC$_FILE_BURST_ONE	
SJC$_FILE_FLAG	SJC$_NO_FILE_FLAG
SJC$_FILE_FLAG_ONE	
SJC$_FILE_TRAILER	SJC$_NO_FILE_TRAILER
SJC$_FILE_TRAILER_ONE	
SJC$_FORM_NAME	
SJC$_FORM_NUMBER	
SJC$_GENERIC_QUEUE	SJC$_NO_GENERIC_QUEUE
SJC$_GENERIC_SELECTION	SJC$_NO_GENERIC_SELECTION
SJC$_GENERIC_TARGET	
SJC$_JOB_BURST	SJC$_NO_JOB_BURST
SJC$_JOB_FLAG	SJC$_NO_JOB_FLAG
SJC$_JOB_LIMIT	
SJC$_JOB_RESET_MODULES	SJC$_NO_JOB_RESET_MODULES
SJC$_JOB_SIZE_MAXIMUM	SJC$_NO_JOB_SIZE_MAXIMUM
SJC$_JOB_SIZE_MINIMUM	SJC$_NO_JOB_SIZE_MINIMUM
SJC$_JOB_SIZE_SCHEDULING	SJC$_NO_JOB_SIZE_SCHEDULING
SJC$_JOB_TRAILER	SJC$_NO_JOB_TRAILER
SJC$_LIBRARY_SPECIFICATION	SJC$_NO_LIBRARY_SPECIFICATION
SJC$_NEXT_JOB	
SJC$_OPEN_QUEUE	
SJC$_OWNER_UIC	
SJC$_PAGINATE	SJC$_NO_PAGINATE
SJC$_PROCESSOR	SJC$_NO_PROCESSOR
SJC$_PROTECTION	
SJC$_QUEUE_DESCRIPTION	SJC$_NO_QUEUE_DESCRIPTION
SJC$_RECORD_BLOCKING	SJC$_NO_RECORD_BLOCKING
SJC$_RELATIVE_PAGE	
SJC$_RETAIN_ALL_JOBS	SJC$_NO_RETAIN_JOBS
SJC$_RETAIN_ERROR_JOBS	
SJC$_SCSNODE_NAME	

SJC$_SEARCH_STRING	
SJC$_SWAP	SJC$_NO_SWAP
SJC$_TERMINAL	SJC$_NO_TERMINAL
SJC$_TOP_OF_FILE	
SJC$_WSDEFAULT	SJC$_NO_WSDEFAULT
SJC$_WSEXTENT	SJC$_NO_WSEXTENT
SJC$_WSQUOTA	SJC$_NO_WSQUOTA

SJC$_START_QUEUE_MANAGER

This request starts the queue manager; it either opens an existing system job queue file or creates a new one. You use the SJC$_QUEUE_FILE_SPECIFICATION item code to specify the name of the job queue file to be used, applying file specification defaults from SYS$SYSTEM:JBCSYSQUE.DAT. Use of the SJC$_NEW_VERSION item code forces the creation of a new system job queue file.

You may specify the following input or Boolean item codes:

SJC$_BUFFER_COUNT	
SJC$_EXTEND_QUANTITY	
SJC$_NEW_VERSION	
SJC$_QUEUE_FILE_SPECIFICATION	
SJC$_QUEMAN_RESTART	SJC$_NO_QUEMAN_RESTART

SJC$_STOP_ACCOUNTING

This request performs two functions. If you specify the SJC$_ACCOUNTING_TYPES item code, the request disables recording of the specified types of accounting records. If you do not specify SJC$_ACCOUNTING_TYPES, the request stops the accounting manager and closes the system accounting file.

You may specify the following input item code:

SJC$_ACCOUNTING_TYPES

SJC$_STOP_QUEUE

This request prevents the starting of jobs in the specified queue. The execution of current jobs is unaffected.

You must specify the following input item code:

SJC$_QUEUE

SJC$_STOP_QUEUE_MANAGER

This request shuts down the queue manager: it stops each queue that is managed by the requesting node; it aborts each job that is currently executing, requeuing those jobs that are restartable; and closes the system job queue file. No item codes are allowed.

SJC$_SYNCHRONIZE_JOB

This request waits for the completion of a job, then sets the event flag, executes the completion AST if you specified **astadr**, and returns the completion status of the job to the I/O Status Block, provided you specified the **iosb** argument.

You must specify the following input item code:

SJC$_QUEUE

You must specify one of the following input item codes:

SJC$_ENTRY_NUMBER
SJC$_JOB_NAME

SJC$_WRITE_ACCOUNTING

This request writes an accounting record.

You must specify the following input item code:

SJC$_ACCOUNTING_MESSAGE

nullarg

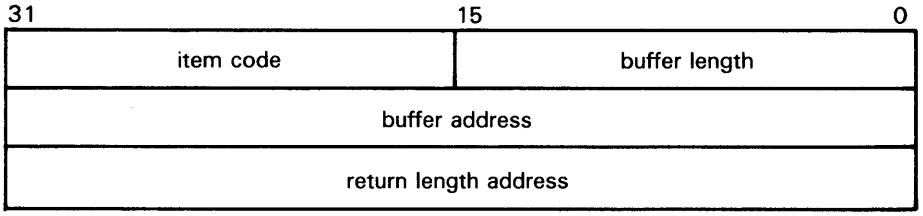
VMS usage: **null_arg**
type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

Place-holding argument reserved by DIGITAL.

itmlst

VMS usage: **item_list_3**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Item list supplying information to be used in performing the function specified by the **func** argument. The **itmlst** argument is the address of the item list. The item list consists of one or more item descriptors, each of which specifies an item code. The item list is terminated by an item code of 0 or by a longword of 0. The following diagram depicts the structure of a single item descriptor.



ZK-1705-84

$SNDJBC Item Descriptor Fields

buffer length

A word specifying the length of the buffer; the buffer either supplies information to be used by $SNDJBC or receives information from $SNDJBC. The required length of the buffer varies depending on the item code specified and is given in the description of each item code.

item code

A word containing an item code, which identifies the nature of the information supplied for use by $SNDJBC or received from $SNDJBC. Each item code has a symbolic name. The $SJCDEF macro defines these symbolic names, which have the following format:

SJC$_code

There are three types of item code:

- Boolean item code. Boolean item codes specify a true or false value: the form SJC$_code specifies a true value; SJC$_NO_code specifies a false value. For Boolean item codes, the **buffer length**, **buffer address**, and **return length** fields of the item descriptor must be zero.
- Input value item code. Input value item codes specify an input value to be used by $SNDJBC. For input value item codes, the **buffer length** and **buffer address** fields of the item descriptor must be nonzero; the **return length** field must be zero. Specific buffer length requirements are given in the description of each item code.
- Output value item code. Output value item codes specify a buffer for information returned by $SNDJBC. For output value item codes, the **buffer length** and **buffer address** fields of the item descriptor must be nonzero; the **return length** field may be zero or nonzero. Specific buffer length requirements are given in the description of each item code.

Several item codes specify a queue name, form name, or characteristic name. For these item codes, the buffer must specify a string containing from 1 to 31 characters, exclusive of spaces, tabs, and null characters, which are ignored. Allowable characters in the string are the uppercase alphabetic characters, the lowercase alphabetic characters (which are converted to uppercase), the numeric characters, the dollar sign ($), and the underscore (_).

buffer address

Address of the buffer that specifies or receives the information.

return length address

Address of a word to receive the length in bytes of information returned by $SNDJBC. If you specify this address as 0, no length is returned.

$SNDJBC Item Codes

SJC$_ACCOUNT_NAME

The SJC$_ACCOUNT_NAME item code is an input value item code. It specifies the account name of the user on behalf of whom the request is made. The buffer must specify a string from one to eight characters. By default, the account name for batch and print jobs is taken from the requesting process.

You need CMKRNL privilege to use this item code.

(Valid for SJC$_CREATE_JOB, SJC$_ENTER_FILE function codes)

SJC$_ACCOUNTING_MESSAGE

The SJC$_ACCOUNTING_MESSAGE item code is an input value item code. It causes the contents of the buffer to be placed in a "user message" accounting record. The buffer must specify a string from 1 to 255 characters.

(Valid for SJC$_WRITE_ACCOUNTING function code)

SJC$_ACCOUNTING_TYPES

The SJC$_ACCOUNTING_TYPES item code is an input value item code. It enables or disables accounting-record types. When an accounting-record type is enabled, the event designated by that type will be recorded in the accounting record. The buffer must contain a longword bit vector wherein each bit set specifies an accounting-record type. Undefined bits must be zero.

The $SJCDEF macro defines the symbolic names for the accounting-record types. Following is a list of each accounting-record type and the system event to which it corresponds.

Accounting-Record Type	Corresponding System Event
SJC$V_ACCT_IMAGE	Image terminations
SJC$V_ACCT_LOGIN_FAILURE	Login failures
SJC$V_ACCT_MESSAGE	User messages sent
SJC$V_ACCT_PRINT	Print job terminations
SJC$V_ACCT_PROCESS	Process terminations

The following accounting-record types, when enabled, provide additional information about image and process termination; specifically, they describe the type of image or process that has terminated.

Accounting-Record Type	Type of Image or Process
SJC$V_ACCT_BATCH	Batch process
SJC$V_ACCT_DETACHED	Detached process
SJC$V_ACCT_INTERACTIVE	Interactive process
SJC$V_ACCT_NETWORK	Network process
SJC$V_ACCT_SUBPROCESS	Subprocess

(Valid for SJC$_START_ACCOUNTING, SJC$_STOP_ACCOUNTING function codes)

SJC$_AFTER_TIME
SJC$_NO_AFTER_TIME

The SJC$_AFTER_TIME item code is an input value item code. It specifies that the job can execute only if the system time is greater than or equal to the quadword time value contained in the buffer. The buffer must contain either an absolute time value or a delta time value; $SNDJBC converts delta time values to absolute time values by adding the current system time.

The SJC$_NO_AFTER_TIME item code is a Boolean item code. It cancels the effect of a SJC$_AFTER_TIME item code previously specified for the job; the job can execute immediately. It is the default.

(Valid for SJC_ALTER_JOB, SJC_CREATE_JOB, SJC$_ENTER_FILE function codes)

SJC$_ALIGNMENT_MASK

The SJC$_ALIGNMENT_MASK item code is a Boolean item code. It is meaningful only for output execution queues and only when the SJC$_ALIGNMENT_PAGES item code is also specified. The SJC$_ALIGNMENT_MASK item code causes the data printed on form

alignment pages to be masked: all alphabetic characters are replaced with the letter "X" and all numeric characters with the number "9".

(Valid for SJC$_START_QUEUE function code)

SJC$_ALIGNMENT_PAGES

The SJC$_ALIGNMENT_PAGES item code is an input value item code. It is meaningful only for output execution queues. It specifies that the queue be placed in form-alignment state and that a number of alignment pages be printed. The buffer must contain a longword value in the range 1 to 20; this value specifies how many alignment pages are to be printed.

(Valid for SJC$_START_QUEUE function code)

SJC$_BASE_PRIORITY

The SJC$_BASE_PRIORITY item code is an input value item code. It is meaningful only for execution queues. It specifies the base priority of batch processes initiated from a batch execution queue or of a symbiont process connected to an output execution queue. A symbiont process can control several queues; however, the base priority of the symbiont process is established by the first queue to which it is connected. The buffer must contain a longword value in the range 0 to 15; this value specifies the base priority.

By default, the base priority is the value of the SYSGEN parameter DEFPRI. If the value of DEFPRI is 0, the default base priority is the base priority of the requesting process.

(Valid for SJC$_ALTER_QUEUE, SJC$_CREATE_QUEUE, SJC$_START_QUEUE function codes)

SJC$_BATCH
SJC$_NO_BATCH

The SJC$_BATCH item code is a Boolean item code. It specifies that the queue is a batch execution queue or a generic batch queue, and thus can process only batch jobs.

The SJC$_BATCH, SJC$_PRINTER, SJC$_SERVER, and SJC$_TERMINAL item codes are mutually exclusive. If none of these item codes are specified, the default is SJC$_PRINTER.

The SJC$_NO_BATCH item code is a Boolean item code. It specifies that the queue is not a batch queue but rather an output execution or generic output queue, and thus can process only print jobs. It is the default.

For the SJC$_START_QUEUE function code, SJC$_BATCH and SJC$_NO_BATCH are supported now for compatibility with VAX/VMS Version 4.n, but may not be supported in the future.

(Valid for SJC$_CREATE_QUEUE, SJC$_START_QUEUE function codes)

SJC$_BUFFER_COUNT

The SJC$_BUFFER_COUNT item code is an input value item code. It specifies the number of buffers that the job controller should allocate to its local buffer cache for performing I/O operations to the system job queue file. The buffer must contain a longword integer value in the range 1 through 127 or 0; this value specifies the number of buffers the job controller allocates to its local buffer cache. If you specify zero, a default value of 50 is used.

(Valid for SJC$_START_QUEUE_MANAGER function code)

SJC$_CHARACTERISTIC_NAME
SJC$_CHARACTERISTIC_NUMBER
SJC$_NO_CHARACTERISTICS

The SJC$_CHARACTERISTIC_NAME and SJC$_CHARACTERISTIC_NUMBER item codes are both input value item codes. Both specify characteristics for jobs or queues, and they may be used interchangeably. The characteristics are user defined.

The SJC$_DEFINE_CHARACTERISTIC and SJC$_DELETE_CHARACTERISTIC function codes include and delete, respectively, a specified characteristic from the system job queue file. A job cannot execute on an execution queue unless the queue possesses all the characteristics possessed by the job; the queue may possess additional characteristics and the job will still execute.

The SJC$_CHARACTERISTIC_NAME and SJC$_CHARACTERISTIC_NUMBER item codes may appear as many times as necessary in a single call to $SNDJBC; the set of characteristics so defined in the call completely replaces the set of characteristics defined by a prior call. The SJC$_NO_CHARACTERISTICS item code cancels all defined characteristics for the job or queue. By default, a queue or job has no defined characteristics.

The string may contain uppercase or lowercase characters (lowercase are converted to uppercase), numeric characters, dollar signs ($), and underscores (_). If the string is a logical name, SYS$SNDJBC translates it iteratively until the equivalence string is found or the number of translations allowed by the system has been performed. The maximum length of the final character string is 31 characters; spaces, tabs, and null characters are ignored.

For SJC$_CHARACTERISTIC_NUMBER, the buffer must contain a longword value in the range 0 to 127. This number identifies a characteristic.

SJC$_NO_CHARACTERISTICS is a Boolean item code.

(For SJC$_CHARACTERISTIC_NAME: Valid for SJC$_ALTER_JOB, SJC$_ALTER_QUEUE, SJC$_CREATE_JOB, SJC$_CREATE_QUEUE, SJC$_DEFINE_CHARACTERISTIC, SJC$_DELETE_CHARACTERISTIC, SJC$_ENTER_FILE, SJC$_START_QUEUE function codes)

(For SJC$_CHARACTERISTIC_NUMBER: Valid for SJC$_ALTER_JOB, SJC$_ALTER_QUEUE, SJC$_CREATE_JOB, SJC$_CREATE_QUEUE, SJC$_DEFINE_CHARACTERISTIC, SJC$_ENTER_FILE, SJC$_START_QUEUE function codes)

SJC$_CHECKPOINT_DATA
SJC$_NO_CHECKPOINT_DATA

The SJC$_CHECKPOINT_DATA item code is an input value item code. It specifies the value of the DCL symbol BATCH$RESTART for a batch job that is restarted. The buffer must contain a string no longer than 255 characters; this string is the value of the symbol BATCH$RESTART.

The SJC$_NO_CHECKPOINT_DATA item code is a Boolean item code. It cancels a previous specification of the BATCH$RESTART symbol; the SJC$_NO_CHECKPOINT_DATA item code also cancels a checkpoint in a print job so that the entire job is reprinted. By default, the BATCH$RESTART symbol is undefined.

(Valid for SJC$_BATCH_CHECKPOINT function code)

SJC$_CLI
SJC$_NO_CLI

The SJC$_CLI item code is an input value item code. It is meaningful only for batch jobs. It specifies that the command language interpreter to be executed is SYS$SYSTEM:name.EXE, where *name* is a valid RMS file name. The buffer must specify a name string from 1 to 39 characters.

The SJC$_NO_CLI item code is a Boolean item code. It specifies that the command language interpreter to be executed is the one specified in the user authorization file. It is the default.

(Valid for SJC$_ALTER_JOB, SJC$_CREATE_JOB, SJC$_ENTER_FILE function codes)

SJC$_CLOSE_QUEUE

The SJC$_CLOSE_QUEUE item code is a Boolean item code. It specifies that jobs cannot be entered in the queue. If the queue is closed, you can specify the SJC$_OPEN_QUEUE item code to permit jobs to be entered in the queue. By default, the queue is open.

Whether a queue is open or closed is independent of any other queue states (such as paused, stalled, stopped).

(Valid for SJC$_ALTER_QUEUE, SJC$_CREATE_QUEUE, SJC$_START_QUEUE function codes)

SJC$_CPU_DEFAULT
SJC$_NO_CPU_DEFAULT

The SJC$_CPU_DEFAULT item code is an input value item code. It is meaningful only for batch execution queues. It specifies the default CPU time limit in 10-millisecond units. The buffer contains this longword value. The value 0 specifies unlimited CPU time. You can specify a value that represents up to 497 days of CPU time.

The SJC$_NO_CPU_DEFAULT item code is a Boolean item code. It is meaningful only for batch execution queues. It specifies that no default CPU time limit is to apply to the job. It is the default.

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. Table SYS–7 shows the action taken when you specify a value for SJC$_CPU_DEFAULT.

Table SYS–7 CPU Time Limit Decision Table

CPU Time Limit Specified for Job?	Default CPU Time Limit Specified for Queue?	Maximum CPU Time Specified for Queue?	Action Taken
No	No	No	Use UAF value
Yes	No	No	Use smaller of job's limit and UAF value
Yes	Yes	No	Use smaller of job's limit and UAF value
Yes	No	Yes	Use smaller of job's limit and maximum
Yes	Yes	Yes	Use smaller of job's limit and maximum
No	Yes	Yes	Use smaller of queue's default and maximum
No	No	Yes	Use maximum
No	Yes	No	Use smaller of UAF value and queue's default

(Valid for SJC$_ALTER_QUEUE, SJC$_CREATE_QUEUE, SJC$_START_QUEUE function codes)

SJC$_CPU_LIMIT
SJC$_NO_CPU_LIMIT

The SJC$_CPU_LIMIT item code is an input value item code. It is meaningful only for batch execution queues and batch jobs. It specifies the maximum CPU time limit for batch jobs in 10 millisecond units. The buffer must contain this longword value. The value 0 specifies unlimited CPU time. You can specify a value that represents up to 497 days of CPU time.

The SJC$_NO_CPU_LIMIT item code is a Boolean item code. It is meaningful only for batch execution queues and batch jobs. It specifies that no maximum CPU time limit is to apply to the job. It is the default.

For information about the action taken when you specify a value for SJC$_CPU_LIMIT, refer to the description of the SJC$_CPU_DEFAULT item code and to Table SYS–7.

(Valid for SJC$_ALTER_JOB, SJC$_ALTER_QUEUE, SJC$_CREATE_JOB, SJC$_CREATE_QUEUE, SJC$_ENTER_FILE, SJC$_START_QUEUE function codes)

SJC$_CREATE_START

The SJC$_CREATE_START item code is a Boolean item code. It specifies that a queue be started after it is created. By default, a queue remains stopped after it is created.

(Valid for SJC$_CREATE_QUEUE function code)

SJC$_DEFAULT_FORM_NAME
SJC$_DEFAULT_FORM_NUMBER

The SJC$_DEFAULT_FORM_NAME and SJC$_DEFAULT_FORM_NUMBER item codes are input value item codes. They specify the default form for a specific output queue by name and by number, respectively.

When you specify a default form for an output queue, the queue uses the queue-specific default form, rather than the systemwide default form, to process any job that does not explicitly specify a form.

For SJC$_DEFAULT_FORM_NAME, the buffer must specify a form name. The string may contain uppercase or lowercase characters (lowercase are converted to uppercase), numeric characters, dollar signs ($), and underscores (_). If the string is a logical name, SYS$SNDJBC translates it iteratively until the equivalence string is found or the number of translations allowed by the system has been performed. The maximum length of the final character string is 31 characters; spaces, tabs, and null characters are ignored.

For SJC$_DEFAULT_FORM_NUMBER, the buffer must specify a longword value. You should use only one of these item codes to identify a default form for the queue.

(Valid for SJC$_ALTER_QUEUE, SJC$_CREATE_QUEUE, SJC$_START_QUEUE function codes)

SJC$_DELETE_FILE
SJC$_NO_DELETE_FILE

The SJC$_DELETE_FILE item code is a Boolean item code. It specifies that a file should be deleted after the job completes. The file that is deleted is the batch or print file submitted for execution. You cannot specify this item code with the SJC$_ALTER_JOB function code, which alters the parameters for an already existing job; you can make a file deletion request only when a job is first submitted to the queue.

The SJC$_NO_DELETE_FILE item code is a Boolean item code. It specifies that a file should not be deleted after execution of the job. It is the default. You can specify this item code with the SJC$_ALTER_JOB function code; this makes it possible to cancel a file deletion request that was made when the job was first submitted to the queue.

(Valid for SJC$_ADD_FILE, SJC$_ENTER_FILE function codes)

SJC$_DESTINATION_QUEUE

The SJC$_DESTINATION_QUEUE item code is an input value item code. When you specify the SJC$_ASSIGN_QUEUE function code, SJC$_DESTINATION_QUEUE specifies the name of the execution queue to which the logical queue is assigned. When you specify the SJC$_ABORT_JOB, SJC$_ALTER_JOB, or SJC$_MERGE_QUEUE function code, SJC$_DESTINATION_QUEUE specifies the name of the queue into which jobs are placed. By default, jobs remain in the original queue.

The string may contain uppercase or lowercase characters (lowercase are converted to uppercase), numeric characters, dollar signs ($), and underscores (_). If the string is a logical name, SYS$SNDJBC translates it iteratively until the equivalence string is found or the number of translations allowed by the system has been performed. The maximum length of the final character string is 31 characters; spaces, tabs, and null characters are ignored.

(Valid for SJC$_ABORT_JOB, SJC$_ALTER_JOB, SJC$_ASSIGN_QUEUE, and SJC$_MERGE_QUEUE function codes)

SJC$_DEVICE_NAME

The SJC$_DEVICE_NAME item code is an input value item code. It specifies the name of the device managed by the output execution queue. The buffer must specify a string from 1 to 31 characters. In a VAXcluster environment, the SJC$_SCSNODE item code is used to specify the name of the node on which the device is located.

(Valid for SJC$_CREATE_QUEUE, SJC$_START_QUEUE function codes)

SJC$_DOUBLE_SPACE
SJC$_NO_DOUBLE_SPACE

The SJC$_DOUBLE_SPACE item code is a Boolean item code. It is meaningful only for output execution queues. It specifies that the symbiont should print the file with double spacing.

The SJC$_NO_DOUBLE_SPACE item code is a Boolean item code. It specifies that the symbiont should print the file with single spacing. It is the default.

(Valid for SJC$_ADD_FILE, SJC$_ALTER_JOB, SJC$_ENTER_FILE function codes)

SJC$_ENTRY_NUMBER

The SJC$_ENTRY_NUMBER item code is an input value item code. It specifies the entry number of the job on which to perform the function. The buffer must contain this entry number.

(Valid for SJC$_ABORT_JOB, SJC$_ALTER_JOB, SJC$_DELETE_JOB, SJC$_SYNCHRONIZE function codes)

SJC$_ENTRY_NUMBER_OUTPUT

The SJC$_ENTRY_NUMBER_OUTPUT item code is an output value item code. The buffer must specify a longword into which $SNDJBC will write the entry number of a created job.

(Valid for SJC$_CREATE_JOB, SJC$_ENTER_FILE function codes)

SJC$_EXTEND_QUANTITY

The SJC$_EXTEND_QUANTITY item code is an input value item code. It specifies the system job queue file extension size in blocks. This extension size is used when the queue file is extended. This value is also used to establish an initial allocation size for the queue file when it is created. The buffer must contain a longword integer value in the range 10 through 65,535, or 0. This value specifies the number of blocks by which the queue should be extended. The default value is 100 blocks. If you specify the value 0, the default size is used.

(Valid for SJC$_START_QUEUE_MANAGER function code)

SJC$_FILE_BURST
SJC$_FILE_BURST_ONE
SJC$_NO_FILE_BURST

The SJC$_FILE_BURST item code is a Boolean item code. It is meaningful only for output execution queues. It specifies that burst and flag pages are to be printed preceding a file. If you specify SJC$_FILE_BURST for a job, it specifies the default for all files in the job; if you specify it for an output execution queue, it specifies the default for all jobs executed from that queue.

The SJC$_FILE_BURST_ONE item code is a Boolean item code. It is meaningful only for output execution queues. It specifies that a burst page is to be printed preceding a file. If you specify SJC$_FILE_BURST_ONE for a job, this item code specifies that a burst page is to be printed preceding only the first copy of the first file in the job; if you specify SJC$_FILE_BURST_ONE for an output execution queue, the item code specifies this behavior as the default for all jobs executed from that queue.

The SJC$_NO_FILE_BURST item code is a Boolean item code. It is meaningful only for output execution queues. It specifies that no burst page should be printed. It is the default.

(For SJC$_FILE_BURST: Valid for SJC$_ADD_FILE, SJC$_ALTER_JOB, SJC$_ALTER_QUEUE, SJC$_CREATE_JOB, SJC$_CREATE_QUEUE, SJC$_ENTER_FILE, SJC$_START_QUEUE function codes)

(For SJC$_FILE_BURST_ONE: Valid for SJC$_ALTER_QUEUE, SJC$_CREATE_JOB, SJC$_CREATE_QUEUE, SJC$_START_QUEUE function codes)

SJC$_FILE_COPIES

The SJC$_FILE_COPIES item code is an input value item code. It is meaningful only for output execution queues. It specifies the number of times a file is printed. By default, a file is repeated once. The buffer must specify a longword value from 1 to 255; this value is the repeat count.

(Valid for SJC$_ADD_FILE, SJC$_ALTER_JOB, SJC$_ENTER_FILE function codes)

SJC$_FILE_FLAG
SJC$_FILE_FLAG_ONE
SJC$_NO_FILE_FLAG

The SJC$_FILE_FLAG item code is a Boolean item code. It is meaningful only for output execution queues. It specifies that a flag page is to be printed preceding a file. If you specify SJC$_FILE_FLAG for a job, this item code indicates the default for all files in the job; if you specify it for an output execution queue, SJC$_FILE_FLAG indicates the default for all jobs executed from that queue.

The SJC$_FILE_FLAG_ONE item code is a Boolean item code. It is meaningful only for output execution queues. It specifies that a flag page is to be printed preceding a file. If you specify SJC$_FILE_FLAG_ONE for a job, this item code specifies that a flag page is to be printed preceding only the first copy of the first file in the job; if you specify SJC$_FILE_FLAG_ONE for an output execution queue, it indicates this behavior as the default for all jobs executed from that queue.

The SJC$_NO_FILE_FLAG item code is a Boolean item code. It is meaningful only for output execution queues. It specifies that no flag page should be printed by default for jobs within the queue.

(For SJC$_FILE_FLAG: Valid for SJC$_ADD_FILE, SJC$_ALTER_JOB, SJC$_ALTER_QUEUE, SJC$_CREATE_JOB, SJC$_CREATE_QUEUE, SJC$_ENTER_FILE, SJC$_START_QUEUE function codes)

(For SJC$_FILE_FLAG_ONE: Valid for SJC$_ALTER_QUEUE, SJC$_CREATE_JOB, SJC$_CREATE_QUEUE, SJC$_START_QUEUE function codes)

SJC$_FILE_IDENTIFICATION

The SJC$_FILE_IDENTIFICATION item code is an input value item code. It specifies the file to be processed. The buffer contains a 28-byte value that identifies the file to be processed. This value specifies (in order) the following three file-identification fields in the RMS NAM block: the 16-byte NAM$T_DVI field, the 6-byte NAM$W_FID field, and the 6-byte NAM$W_DID field. These fields occur consecutively, in the NAM block.

If you specify SJC$_FILE_IDENTIFICATION, you must omit the SJC$_FILE_SPECIFICATION item code.

(Valid for SJC$_ADD_FILE, SJC$_ENTER_FILE function codes)

SJC$_FILE_SETUP_MODULES
SJC$_NO_FILE_SETUP_MODULES

The SJC$_FILE_SETUP_MODULES item code is an input value item code. It is meaningful only for output execution queues. It specifies that a list of text modules should be extracted from the device control library and copied to the printer before a file is printed. The buffer must contain a list of text module names, with a comma separating each name.

The SJC$_NO_FILE_SETUP_MODULES item code is a Boolean item code. It is meaningful only for output execution queues. It specifies that no text modules should be copied before printing a file. It is the default.

(Valid for SJC$_ADD_FILE, SJC$_ALTER_JOB, SJC$_ENTER_FILE function codes)

SJC$_FILE_SPECIFICATION

The SJC$_FILE_SPECIFICATION item code is an input value item code. It identifies the file to be processed. The buffer must contain the file specification of the file to be processed. The $SNDJBC service converts the file specification to the corresponding file identification and proceeds as if the SJC$_FILE_IDENTIFICATION item code had been specified. If you specify SJC$_FILE_SPECIFICATION, you must omit the SJC$_FILE_IDENTIFICATION item code.

(Valid for SJC$_ADD_FILE, SJC$_ENTER_FILE function codes)

SJC$_FILE_TRAILER
SJC$_FILE_TRAILER_ONE
SJC$_NO_FILE_TRAILER

The SJC$_FILE_TRAILER item code is a Boolean item code. It is meaningful only for output execution queues. It specifies that a trailer page is to be printed following a file. If you specify SJC$_FILE_TRAILER for a job, this item code indicates the default for all files in the job; if you specify it for an output execution queue, SJC$_FILE_TRAILER specifies the default for all jobs executed on that queue.

The SJC$_FILE_TRAILER_ONE item code is a Boolean item code. It is meaningful only for output execution queues. It specifies that a trailer page is to be printed following a file. If you specify SJC$_FILE_TRAILER_ONE for a job, this item code indicates that a trailer page is to be printed following only the last copy of the last file in the job; if you specify it for an output execution queue, SJC$_FILE_TRAILER_ONE indicates this behavior as the default for all jobs executed on that queue.

The SJC$_NO_FILE_TRAILER item code is a Boolean item code. It is meaningful only for output execution queues. It specifies that no trailer page should be printed. It is the default.

(For SJC$_FILE_TRAILER: Valid for SJC$_ADD_FILE, SJC$_ALTER_JOB, SJC$_ALTER_QUEUE, SJC$_CREATE_JOB, SJC$_CREATE_QUEUE, SJC$_ENTER_FILE, SJC$_START_QUEUE function codes)

(For SJC$_FILE_TRAILER_ONE: Valid for SJC$_ALTER_QUEUE, SJC$_CREATE_JOB, SJC$_CREATE_QUEUE, SJC$_START_QUEUE function codes)

SJC$_FIRST_PAGE
SJC$_NO_FIRST_PAGE

The SJC$_FIRST_PAGE item code is an input value item code. It is meaningful only for jobs queued to output execution queues. It specifies the page number at which printing should begin. The buffer must contain a nonzero longword value specifying this page number.

The SJC$_NO_FIRST_PAGE item code is a Boolean item code. It is meaningful only for jobs queued to output execution queues. It specifies that printing should begin with the first page. It is the default.

(Valid for SJC$_ADD_FILE, SJC$_ALTER_JOB, SJC$_ENTER_FILE function codes)

SJC$_FORM_DESCRIPTION

The SJC$_FORM_DESCRIPTION item code is an input value item code. It provides operator-supplied information describing the form. By default, the form name is used. The buffer must specify a string of no more than 255 characters.

(Valid for SJC$_DEFINE_FORM function code)

SJC$_FORM_LENGTH

The SJC$_FORM_LENGTH item code is an input value item code. It specifies the physical length of the form in lines. The buffer must contain a nonzero longword integer value. By default, the form length is 66 lines.

(Valid for SJC$_DEFINE_FORM function code)

SJC$_FORM_MARGIN_BOTTOM

The SJC$_FORM_MARGIN_BOTTOM item code is an input value item code. It specifies the bottom margin of the form in lines. By default, the bottom margin is 6 lines.

(Valid for SJC$_DEFINE_FORM function code)

SJC$_FORM_MARGIN_LEFT

The SJC$_FORM_MARGIN_LEFT item code is an input value item code. It specifies the width of the left margin of the form in characters. By default, the left margin is 0. The buffer must specify a longword value.

(Valid for SJC$_DEFINE_FORM function code)

SJC$_FORM_MARGIN_RIGHT

The SJC$_FORM_MARGIN_RIGHT item code is an input value item code. It specifies the width of the right margin of the form in characters. By default, the right margin is 0. The buffer must specify a longword value.

(Valid for SJC$_DEFINE_FORM function code)

SJC$_FORM_MARGIN_TOP

The SJC$_FORM_MARGIN_TOP item code is an input value item code. It specifies the top margin of the form in lines. By default, the top margin is 0.

(Valid for SJC$_DEFINE_FORM function code)

SJC$_FORM_NAME
SJC$_FORM_NUMBER

The SJC$_FORM_NAME and SJC$_FORM_NUMBER item codes are input value item codes. They specify a mounted form for the queue by name and by number, respectively. For SJC$_FORM_NAME, the buffer must specify a form name. For SJC$_FORM_NUMBER, the buffer must specify a longword value. You should use only one of these two item codes to identify a form in queue- and job-related function codes.

The SJC$_DEFINE_FORM and SJC$_DELETE_FORM function codes include and delete a specified form name and number, respectively, from the system job queue file. The mounted form indicates the stock type of the output queue. A job cannot execute on an output queue unless the stock type of the form specified (by name or number) for the job item code is the same as the stock type of the mounted form specified for the queue. For more information about how the stock type of a form affects job processing, see the *Guide to Maintaining a VMS System*.

The string may contain uppercase or lowercase characters (lowercase are converted to uppercase), numeric characters, dollar signs ($), and underscores (_). If the string is a logical name, SYS$SNDJBC translates it iteratively until the equivalence string is found or the number of translations allowed by the system has been performed. The maximum length of the final character string is 31 characters; spaces, tabs, and null characters are ignored.

(For SJC$_FORM_NAME: Valid for SJC$_ALTER_JOB, SJC$_ALTER_QUEUE, SJC$_CREATE_JOB, SJC$_CREATE_QUEUE, SJC$_DEFINE_FORM, SJC$_DELETE_FORM, SJC$_ENTER_FILE, SJC$_START_QUEUE function codes)

(For SJC$_FORM_NUMBER: Valid for SJC$_ALTER_JOB, SJC$_ALTER_QUEUE, SJC$_CREATE_JOB, SJC$_CREATE_QUEUE, SJC$_DEFINE_FORM, SJC$_ENTER_FILE, SJC$_START_QUEUE function codes)

SJC$_FORM_SETUP_MODULES
SJC$_NO_FORM_SETUP_MODULES

The SJC$_FORM_SETUP_MODULES item code is an input value item code. The buffer must specify one or more text module names, with a comma separating each name. This item code specifies that these modules should be extracted from the device control library and copied to the printer before each file that is printed on the form.

The SJC$_NO_FORM_SETUP_MODULES item code is a Boolean item code. It specifies that no device control modules should be copied. It is the default.

(Valid for SJC$_DEFINE_FORM function code)

SJC$_FORM_SHEET_FEED
SJC$_NO_FORM_SHEET_FEED

The SJC$_FORM_SHEET_FEED item code is a Boolean item code. It specifies that the symbiont should pause at the end of each physical page so that a new sheet may be inserted.

The SJC$_NO_FORM_SHEET_FEED item code is a Boolean item code. It specifies that the output symbiont should not pause at the end of every physical page. It is the default.

(Valid for SJC$_DEFINE_FORM function code)

SJC$_FORM_STOCK

The SJC$_FORM_STOCK item code is an input value item code. It specifies a name for the paper stock. The buffer must contain a string of 1 to 31 characters. By default, the name of the paper stock is the form name.

(Valid for SJC$_DEFINE_FORM function code)

SJC$_FORM_TRUNCATE
SJC$_NO_FORM_TRUNCATE

The SJC$_FORM_TRUNCATE item code is a Boolean item code. It specifies that the symbiont should truncate lines that extend beyond the right margin. Specifying SJC$_FORM_TRUNCATE cancels SJC$_FORM_WRAP. The SJC$_FORM_TRUNCATE item code is the default.

The SJC$_NO_FORM_TRUNCATE item code is a Boolean item code. It specifies that the output symbiont should not truncate lines that extend beyond the right margin.

(Valid for SJC$_DEFINE_FORM function code)

SJC$_FORM_WIDTH

The SJC$_FORM_WIDTH item code is an input value item code. It specifies the physical width of the form in characters. The buffer must contain a nonzero longword integer. By default, the form width is 132 characters.

SJC$_FORM_WRAP
SJC$_NO_FORM_WRAP

The SJC$_FORM_WRAP item code is a Boolean item code. It specifies that the symbiont should wrap lines that extend beyond the right margin. Specifying SJC$_FORM_WRAP cancels SJC$_FORM_TRUNCATE.

The SJC$_NO_FORM_WRAP item code is a Boolean item code. It specifies that the output symbiont should not wrap lines. It is the default.

(Valid for SJC$_DEFINE_FORM function code)

SJC$_GENERIC_QUEUE
SJC$_NO_GENERIC_QUEUE

The SJC$_GENERIC_QUEUE item code is a Boolean item code. It specifies that a queue is a generic queue.

The SJC$_NO_GENERIC_QUEUE item code is a Boolean item code. It specifies that a queue is not a generic queue. It is the default. By default, a queue is an execution queue; see the Description section for a full discussion of the types of queue.

(Valid for SJC$_CREATE_QUEUE, SJC$_START_QUEUE function codes)

SJC$_GENERIC_SELECTION
SJC$_NO_GENERIC_SELECTION

The SJC$_GENERIC_SELECTION item code is a Boolean item code. It specifies that an execution queue can accept jobs from a generic queue. It is the default. It is meaningful only for execution queues.

The SJC$_NO_GENERIC_SELECTION item code is a Boolean item code. It specifies that an execution queue cannot accept jobs from a generic queue.

(Valid for SJC$_ALTER_QUEUE, SJC$_CREATE_QUEUE, SJC$_START_QUEUE function codes)

SJC$_GENERIC_TARGET

The SJC$_GENERIC_TARGET item code is an input value item code. The buffer must specify a queue name. This queue name identifies an execution queue that can accept jobs from a generic queue. This item code is meaningful only for generic queues.

This item code can appear up to 124 times in a single call to $SNDJBC. The set of queues defined in a single call completely replaces the set defined by a prior call.

The string may contain uppercase or lowercase characters (lowercase are converted to uppercase), numeric characters, dollar signs ($), and underscores (_). If the string is a logical name, SYS$SNDJBC translates it iteratively until the equivalence string is found or the number of translations allowed by the system has been performed. The maximum length of the final character string is 31 characters; spaces, tabs, and null characters are ignored.

(Valid for SJC$_CREATE_QUEUE, SJC$_START_QUEUE function codes)

SJC$_HOLD
SJC$_NO_HOLD

The SJC$_HOLD item code is a Boolean item code. It specifies that a job cannot execute and must enter a holding status.

The SJC$_NO_HOLD item code is a Boolean item code. It specifies that a job can execute immediately when used with the SJC$_ALTER_JOB function code. It makes the following types of job eligible for execution: (1) a job that is holding because it was specified with the SJC$_HOLD item code, (2) a job that was refused by the symbiont, and (3) a job that was retained after

execution. It is the default. SJC$_NO_HOLD does not release a job that is specified with the SJC$_AFTER_TIME item code.

(Valid for SJC$_ABORT_JOB, SJC$_ALTER_JOB, SJC$_CREATE_JOB, SJC$_ENTER_FILE function codes)

SJC$_JOB_BURST
SJC$_NO_JOB_BURST

The SJC$_JOB_BURST item code is a Boolean item code. It specifies that burst and flag pages are to be printed preceding each job. It is meaningful only for output execution queues.

The SJC$_NO_JOB_BURST item code is a Boolean item code. It specifies that a burst page is not to be printed preceding each job. It is meaningful only for output execution queues. It is the default.

(Valid for SJC$_ALTER_QUEUE, SJC$_CREATE_QUEUE, SJC$_START_QUEUE function codes)

SJC$_JOB_COPIES

The SJC$_JOB_COPIES item code is an input value item code. It specifies the number of times that the entire print job is to be repeated. The buffer must contain this nonzero longword integer value. By default, the print job is repeated once.

(Valid for SJC$_ALTER_JOB, SJC$_CREATE_JOB, SJC$_ENTER_FILE function codes)

SJC$_JOB_FLAG
SJC$_NO_JOB_FLAG

The SJC$_JOB_FLAG item code is a Boolean item code. It specifies that a flag page is to be printed preceding each job. It is meaningful only for output execution queues.

The SJC$_NO_JOB_FLAG item code is a Boolean item code. It specifies that a flag page is not to be printed preceding each job. It is meaningful only for output execution queues. It is the default.

(Valid for SJC$_ALTER_QUEUE, SJC$_CREATE_QUEUE, SJC$_START_QUEUE function codes)

SJC$_JOB_LIMIT

The SJC$_JOB_LIMIT item code is an input value item code. It specifies the maximum number of jobs that can execute simultaneously on a queue. The buffer must contain a longword value in the range 1 to 255. It is meaningful only for batch execution queues. By default, the job limit is 1.

(Valid for SJC$_ALTER_QUEUE, SJC$_CREATE_QUEUE, SJC$_START_QUEUE function codes)

SJC$_JOB_NAME

The SJC$_JOB_NAME item code is an input value item code. It specifies the name of a job. The buffer must specify a string from 1 to 39 characters.

For function codes SJC$_ENTER_FILE, SJC$_CREATE_JOB, and SJC$_ALTER_JOB, SJC$_JOB_NAME specifies the identifying name of the job. By default, the name used is the name of the first file in the job.

For function code SJC$_SYNCHRONIZE_JOB, SJC$_JOB_NAME specifies the name of the job on which to operate. The job name is implicitly qualified by the user name.

(Valid for SJC$_ALTER_JOB, SJC$_CREATE_JOB, SJC$_ENTER_FILE, SJC$_SYNCHRONIZE function codes)

SJC$_JOB_RESET_MODULES
SJC$_NO_JOB_RESET_MODULES

The SJC$_JOB_RESET_MODULES item code is an input value item code. It is meaningful only for output execution queues. The buffer must specify the names of one or more text modules, with a comma separating each name. This item code specifies that these modules are to be extracted from the device control library and copied to the printer before each print job.

The SJC$_NO_JOB_RESET_MODULES item code is a Boolean item code. It specifies that no text modules should be copied to the printer. It is the default.

(Valid for SJC$_ALTER_QUEUE, SJC$_CREATE_QUEUE, SJC$_START_QUEUE function codes)

SJC$_JOB_SIZE_MAXIMUM
SJC$_NO_JOB_SIZE_MAXIMUM

The SJC$_JOB_SIZE_MAXIMUM item code is an input value item code. It is meaningful only for output execution queues. It specifies that a print job can execute only if its total size in blocks is less than or equal to the specified value. The buffer specifies this nonzero longword value.

The SJC$_NO_JOB_SIZE_MAXIMUM item code is a Boolean item code. It specifies that a print job can execute immediately regardless of its size. It is the default.

(Valid for SJC$_ALTER_QUEUE, SJC$_CREATE_QUEUE, SJC$_START_QUEUE function codes)

SJC$_JOB_SIZE_MINIMUM
SJC$_NO_JOB_SIZE_MINIMUM

The SJC$_JOB_SIZE_MINIMUM item code is an input value item code. It is meaningful only for output execution queues. It specifies that a print job can execute only if its total size in blocks is greater than or equal to the specified value. The buffer specifies this nonzero longword value.

The SJC$_NO_JOB_SIZE_MINIMUM item code is a Boolean item code. It specifies that a print job can execute immediately regardless of its size. It is the default.

(Valid for SJC$_ALTER_QUEUE, SJC$_CREATE_QUEUE, SJC$_START_QUEUE function codes)

SJC$_JOB_SIZE_SCHEDULING
SJC$_NO_JOB_SIZE_SCHEDULING

The SJC$_JOB_SIZE_SCHEDULING item code is a Boolean item code. It specifies that print jobs entered in an output execution queue should be scheduled according to size, with the smallest job of a given priority processed first. It is the default.

The SJC$_NO_JOB_SIZE_SCHEDULING item code is a Boolean item code. It specifies that print jobs of a given priority should not be scheduled according to print size.

Changing the value of this item code for a queue while print jobs are pending on any queue produces unpredictable results.

(Valid for SJC$_ALTER_QUEUE, SJC$_CREATE_QUEUE, SJC$_START_QUEUE function codes)

SJC$_JOB_STATUS_OUTPUT

The SJC$_JOB_STATUS_OUTPUT item code is an output value item code. When specified, $SNDJBC returns, as a character string, a textual message describing the status of a submitted job. Because the message can include up to 255 characters, the buffer length field of the item descriptor should specify 255 (bytes).

(Valid for SJC$_CLOSE_JOB, SJC$_ENTER_FILE function codes)

SJC$_JOB_TRAILER
SJC$_NO_JOB_TRAILER

The SJC$_JOB_TRAILER item code is a Boolean item code. It is meaningful only for output execution queues. It specifies that a trailer page is to be printed following each job.

The SJC$_NO_JOB_TRAILER item code is a Boolean item code. It is meaningful only for output execution queues. It specifies that a trailer page is not to be printed following each job. It is the default.

(Valid for SJC$_ALTER_QUEUE, SJC$_CREATE_QUEUE, SJC$_START_QUEUE function codes)

SJC$_LAST_PAGE
SJC$_NO_LAST_PAGE

The SJC$_LAST_PAGE item code is an input value item code. It is meaningful only for jobs submitted to output execution queues. It specifies the page number at which printing should end. The buffer specifies this nonzero longword value.

The SJC$_NO_LAST_PAGE item code is a Boolean item code. It specifies that printing should end after the last page. It is the default.

(Valid for SJC$_ADD_FILE, SJC$_ALTER_JOB, SJC$_ENTER_FILE function codes)

SJC$_LIBRARY_SPECIFICATION
SJC$_NO_LIBRARY_SPECIFICATION

The SJC$_LIBRARY_SPECIFICATION item code is an input value item code. It is meaningful only for output execution queues. It specifies that the device control library for the queue is SYS$LIBRARY:name.TLB, where *name* is a valid RMS file name. The buffer must specify the RMS file name.

The SJC$_NO_LIBRARY_SPECIFICATION item code is a Boolean item code. It specifies that the device control library is SYS$LIBRARY:SYSDEVCTL.TLB. It is the default.

(Valid for SJC$_CREATE_QUEUE, SJC$_START_QUEUE function codes)

SJC$_LOG_DELETE
SJC$_NO_LOG_DELETE

The SJC$_LOG_DELETE item code is a Boolean item code. It specifies that the log file produced for a batch job is to be deleted. It is meaningful only for batch jobs. It is the default.

The SJC$_NO_LOG_DELETE item code is a Boolean item code. It specifies that the log file produced for a batch job is not to be deleted.

(Valid for SJC$_ALTER_JOB, SJC$_CREATE_JOB, SJC$_ENTER_FILE function codes)

SJC$_LOG_QUEUE

The SJC$_LOG_QUEUE item code is an input value item code. It is meaningful only for batch jobs. It specifies the queue into which the log file produced for the batch job is entered for printing. The buffer must specify the name of the queue. By default, the log file is entered in queue SYS$PRINT.

The string may contain uppercase or lowercase characters (lowercase are converted to uppercase), numeric characters, dollar signs ($), and underscores (_). If the string is a logical name, SYS$SNDJBC translates it iteratively until the equivalence string is found or the number of translations allowed by the system has been performed. The maximum length of the final character string is 31 characters; spaces, tabs, and null characters are ignored.

(Valid for SJC$_ALTER_JOB, SJC$_CREATE_JOB, SJC$_ENTER_FILE function codes)

SJC$_LOG_SPECIFICATION
SJC$_NO_LOG_SPECIFICATION

The SJC$_LOG_SPECIFICATION item code is an input value item code. It is meaningful only for batch jobs. It specifies the file specification of the log file produced for a batch job. The buffer must contain this RMS file specification. Omitted fields in the file specification are supplied from the default file specification SYS$LOGIN:name.LOG, where *name* is the job name. By default a log file is produced using this default file specification to generate the log file name.

The SJC$_NO_LOG_SPECIFICATION item code is a Boolean item code. It specifies that no log file should be produced for the batch job.

(Valid for SJC$_ALTER_JOB, SJC$_CREATE_JOB, SJC$_ENTER_FILE function codes)

SJC$_LOG_SPOOL
SJC$_NO_LOG_SPOOL

The SJC$_LOG_SPOOL item code is a Boolean item code. It specifies that the log file produced for a batch job is to be printed. It is meaningful only for batch jobs. It is the default.

The SJC$_NO_LOG_SPOOL item code is a Boolean item code. It specifies that the log file for a batch job is not to be printed.

(Valid for SJC$_ALTER_JOB, SJC$_CREATE_JOB, SJC$_ENTER_FILE function codes)

SJC$_LOWERCASE
SJC$_NO_LOWERCASE

The SJC$_LOWERCASE item code is a Boolean item code. It specifies that a job can execute only on a device that has the LOWERCASE device-dependent characteristic. It is meaningful only for jobs submitted to output execution queues.

The SJC$_NO_LOWERCASE item code is a Boolean item code. It specifies that a job can execute whether or not the output device has the LOWERCASE device-dependent characteristic. It is the default.

(Valid for SJC$_ALTER_JOB, SJC$_CREATE_JOB, SJC$_ENTER_FILE function codes)

SJC$_NEW_VERSION

The SJC$_NEW_VERSION item code is a Boolean item code. It specifies that a new version of the system job queue file or system accounting file is to be created, whether or not the file already exists. By default, the system job queue file or accounting file is created only if it does not already exist.

(Valid for SJC$_START_ACCOUNTING, SJC$_START_QUEUE_MANAGER function codes)

SJC$_NEXT_JOB

The SJC$_NEXT_JOB item code is a Boolean item code. It is meaningful only for output execution queues. It specifies that the current job should be aborted and that printing should be resumed with the next job.

(Valid for SJC$_START_QUEUE function code)

SJC$_NOTE
SJC$_NO_NOTE

The SJC$_NOTE item code is an input value item code. It is meaningful only for output execution queues. It specifies a string to be printed on the job flag and file flag pages. The buffer must specify this string.

The SJC$_NO_NOTE item code is a Boolean item code. It specifies that no string is to be printed on the job flag and file flag pages. It is the default.

(Valid for SJC$_ALTER_JOB, SJC$_CREATE_JOB, SJC$_ENTER_FILE function codes)

SJC$_NOTIFY
SJC$_NO_NOTIFY

The SJC$_NOTIFY item code is a Boolean item code. It specifies that a message is to be broadcast, at the time of job completion, to each logged-in terminal, of the user who submitted the job.

The SJC$_NO_NOTIFY item code is a Boolean item code. It specifies that no message is to be broadcast at the time of job completion. It is the default.

(Valid for SJC$_ALTER_JOB, SJC$_CREATE_JOB, SJC$_ENTER_FILE function codes)

SJC$_OPEN_QUEUE

The SJC$_OPEN_QUEUE item code is a Boolean item code. It specifies that jobs can be entered in the queue. To specify that jobs cannot be entered in the queue, use the SJC$_CLOSE_QUEUE item code. By default, the queue is open.

Whether a queue is open or closed is independent of any other queue states (such as paused, stalled, stopped).

(Valid for SJC$_ALTER_QUEUE, SJC$_CREATE_QUEUE, SJC$_START_QUEUE function codes)

SJC$_OPERATOR_REQUEST
SJC$_NO_OPERATOR_REQUEST

The SJC$_OPERATOR_REQUEST item code is an input value item code. It is meaningful only for output execution queues. The buffer must contain a text string. This item code specifies that when a job begins execution, the execution queue is to be placed in the paused state and the specified text string is to be included in a message to the queue operator requesting service.

The SJC$_NO_OPERATOR_REQUEST item code is a Boolean item code. It specifies that no message is to be sent to the queue operator. It is the default.

(Valid for SJC$_ALTER_JOB, SJC$_CREATE_JOB, SJC$_ENTER_FILE function codes)

SJC$_OWNER_UIC

The SJC$_OWNER_UIC item code is an input value item code. It specifies the owner UIC of a queue. The buffer must specify the longword UIC. By default, the owner UIC is [1,4].

(Valid for SJC$_ALTER_QUEUE, SJC$_CREATE_QUEUE, SJC$_START_QUEUE function codes)

SJC$_PAGE_HEADER
SJC$_NO_PAGE_HEADER

The SJC$_PAGE_HEADER item code is a Boolean item code. It is meaningful only for output execution queues. It specifies that a page heading is to be printed on each page of output.

The SJC$_NO_PAGE_HEADER item code is a Boolean item code. It specifies that no page heading is to be printed. It is the default.

(Valid for SJC$_ADD_FILE, SJC$_ALTER_JOB, SJC$_ENTER_FILE function codes)

SJC$_PAGE_SETUP_MODULES
SJC$_NO_PAGE_SETUP_MODULES

The SJC$_PAGE_SETUP_MODULES item code is an input value item code. The buffer must specify one or more text module names, with a comma separating each name. This item code specifies that these modules are to be extracted from the device control library and copied to the printer before each page is printed.

The SJC$_NO_PAGE_SETUP_MODULES item code is a Boolean item code. It specifies that no device control modules are to be copied. It is the default.

(Valid for SJC$_DEFINE_FORM function code)

SJC$_PAGINATE
SJC$_NO_PAGINATE

The SJC$_PAGINATE item code is a Boolean item code. It is meaningful only for output execution queues and jobs submitted to output execution queues. It specifies that the symbiont should paginate the output by inserting

a form feed whenever output reaches the bottom margin of the form. It is the default.

The SJC$_NO_PAGINATE item code is a Boolean item code. It specifies that the symbiont should not paginate the output.

(Valid for SJC$_ADD_FILE, SJC$_ALTER_JOB, SJC$_ALTER_QUEUE, SJC$_CREATE_QUEUE, SJC$_ENTER_FILE, SJC$_START_QUEUE function codes)

SJC$_PARAMETER_1 through SJC$_PARAMETER_8 SJC$_NO_PARAMETERS

The SJC$_PARAMETER_1 through SJC$_PARAMETER_8 item codes are input value item codes; the last digit of the item code name is a number from 1 through 8. For each item code specified, the buffer must specify a string of no more than 255 characters. For batch jobs, the string becomes the value of the DCL symbol P1 through P8, respectively, within the outermost command procedure.

For print jobs, the system makes the string available to the symbiont, though the standard VMS print symbiont does not use this information. By default, each of the eight parameters specifies a null string.

For function code SJC$_ALTER_JOB, if any SJC$_PARAMETER item is specified, the value of each unspecified item is the null string.

The SJC$_NO_PARAMETERS item code is a Boolean item code. It specifies that none of the SJC$_PARAMETER items are to be passed in the batch or print job. It is the default.

(Valid for SJC$_ALTER_JOB, SJC$_CREATE_JOB, SJC$_ENTER_FILE function codes)

SJC$_PASSALL SJC$_NO_PASSALL

The SJC$_PASSALL item code is a Boolean item code. It is meaningful only for jobs submitted to output execution queues. It specifies that the symbiont is to print the file in PASSALL mode.

The SJC$_NO_PASSALL item code is a Boolean item code. It specifies that the symbiont is not to print the file in PASSALL mode. It is the default.

(Valid for SJC$_ADD_FILE, SJC$_ALTER_JOB, SJC$_ENTER_FILE function codes)

SJC$_PRINTER

The SJC$_PRINTER item code is a Boolean item code. It is meaningful only for output queues. It specifies that the queue being created is a printer queue. The SJC$_BATCH, SJC$_PRINTER, SJC$_SERVER, and SJC$_TERMINAL item codes are mutually exclusive. If none of these item codes are specified, the default is SJC$_PRINTER.

(Valid for SJC$_CREATE_QUEUE function code)

SJC$_PRIORITY

The SJC$_PRIORITY item code is an input value item code. The buffer must specify a longword value in the range 0 through 255. This value specifies the scheduling priority of the job in a queue relative to the scheduling priority of other jobs in the same queue.

By default, the scheduling priority of the job is the value of the SYSGEN parameter DEFQUEPRI. If the value of DEFQUEPRI is 0, the default scheduling priority is the base priority of the requesting process.

If you specify a value for SJC$_PRIORITY that is greater than the SYSGEN parameter MAXQUEPRI and you do not have either ALTPRI or OPER privilege, the system uses the value of MAXQUEPRI. If you have either ALTPRI or OPER privilege, the system uses any value you specify for SJC$_PRIORITY, even if it is included in the range between MAXQUEPRI + 1 and 255.

(Valid for SJC$_ABORT_JOB, SJC$_ALTER_JOB, SJC$_CREATE_JOB, SJC$_ENTER_FILE function codes)

SJC$_PROCESSOR
SJC$_NO_PROCESSOR

The SJC$_PROCESSOR item code is an input value item code. The buffer must specify a valid RMS file name.

When specified for an output execution queue, SJC$_PROCESSOR specifies that the symbiont image to be executed is SYS$SYSTEM:name.EXE, where *name* is the RMS file name contained in the buffer.

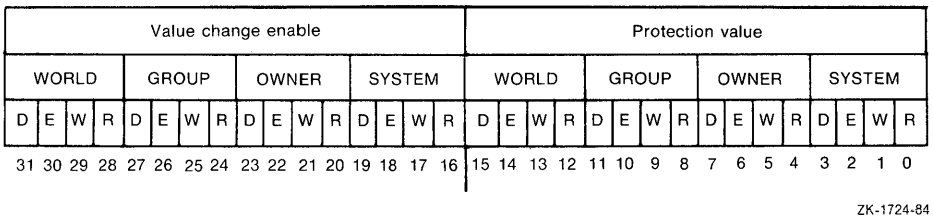
When specified for a generic output queue, SJC$_PROCESSOR specifies that the generic queue can place jobs only in server queues that are executing the symbiont image SYS$SYSTEM:name.EXE, where *name* is the RMS file name contained in the buffer.

The SJC$_NO_PROCESSOR item code is a Boolean item code. It specifies that the symbiont image to be executed is SYS$SYSTEM:PRTSMB.EXE. It is the default.

(Valid for SJC$_CREATE_QUEUE, SJC$_START_QUEUE function codes)

SJC$_PROTECTION

SJC$_PROTECTION is an input value item code. It specifies the protection of a queue. The buffer must specify a longword in the format shown in the following diagram.



ZK-1724-84

Bits 0 through 15 specify the protection value: the four types of access (read, write, execute, delete) to be granted to the four classes of user (system, owner, group, world). Set bits deny access and clear bits allow access.

Bits 16 through 31 enable or disable the interpretation of bits 0 through 15. When a bit in the second word is set, the corresponding bit in the first word will affect the queue protection. When a bit in the second word is clear, the corresponding bit in the first word is ignored.

By default, the queue protection is (S:E,O:D,G:R,W:W).

Note that you can assign ACLs to queues using the $CHANGE_ACL system service.

(Valid for SJC$_ALTER_QUEUE, SJC$_CREATE_QUEUE, SJC$_START_QUEUE function codes)

SJC$_QUEMAN_RESTART
SJC$_NO_QUEMAN_RESTART

The SJC$_QUEMAN_RESTART item code is a Boolean item code. It specifies that the job controller should automatically restart the queue manager when the job controller recovers from an internal fatal error. An internal fatal error causes the job controller to close the job queue manager file and stop the queue manager.

If you specify SJC$_QUEMAN_RESTART, batch and output queues are restored to the states that existed prior to the job controller failure. The job controller opens the job queue manager file that was open when the job controller aborted. The system uses the default values of 100 for SJC$_EXTEND_QUANTITY and 50 for SJC$_BUFFER_COUNT, rather than the values you may have specified for these item codes when you last started the queue manager with a SJC$_START_QUEUE_MANAGER operation.

Note: To prevent a looping condition, the job controller does not restart the queue manager if it detects a job controller error within 2 minutes of starting the queue manager. This algorithm may change in a future release of VMS.

The SJC$_NO_QUEMAN_RESTART item code is a Boolean item code. It specifies that the job controller should not restart the queue manager when it recovers from an internal job controller fatal error. In this case, a user with OPER privilege must restart the queue manager and restore the queuing environment. It is the default.

(Valid for SJC$_START_QUEUE_MANAGER function code)

SJC$_QUEUE

The SJC$_QUEUE item code is an input value item code. It specifies the queue to which the operation is directed. The buffer must specify the name of the queue.

The string may contain uppercase or lowercase characters (lowercase are converted to uppercase), numeric characters, dollar signs ($), and underscores(_). If the string is a logical name, SYS$SNDJBC translates it iteratively until the equivalence string is found or the maximum number of translations allowed by the system has been performed. The maximum length of the final character string is 31 characters; spaces, tabs, and null characters are ignored.

(Valid for SJC$_ABORT_JOB, SJC$_ALTER_JOB, SJC$_ALTER_QUEUE, SJC$_CREATE_JOB, SJC$_CREATE_QUEUE, SJC$_DELETE_JOB, SJC$_DELETE_QUEUE, SJC$_ENTER_FILE, SJC$_START_QUEUE, SJC$_SYNCHRONIZE function codes)

SJC$_QUEUE_DESCRIPTION
SJC$_NO_QUEUE_DESCRIPTION

The SJC$_QUEUE_DESCRIPTION item code is an input value item code. It provides operator-supplied information about the queue. The buffer must specify a string of no more than 255 characters.

The SJC$_NO_QUEUE_DESCRIPTION item code is a Boolean item code. It specifies that no description is associated with the queue.

(Valid for SJC$_ALTER_QUEUE, SJC$_CREATE_QUEUE, SJC$_START_QUEUE function codes)

SJC$_QUEUE_FILE_SPECIFICATION

The SJC$_QUEUE_FILE_SPECIFICATION item code is an input value item code. It specifies the file specification of the system job queue file. The buffer must contain a valid RMS file specification. Omitted fields in the file specification are supplied from the default file specification SYS$SYSTEM:JBCSYSQUE.DAT.

(Valid for SJC$_START_QUEUE_MANAGER function code)

SJC$_RECORD_BLOCKING
SJC$_NO_RECORD_BLOCKING

The SJC$_RECORD_BLOCKING item code is a Boolean item code. It is meaningful only for output execution queues. It specifies that the symbiont can merge the output records it sends to the output device into a single I/O request. For the standard VMS print symbiont, record blocking can have a significant performance advantage over single-record mode. It is the default.

The SJC$_NO_RECORD_BLOCKING item code is a Boolean item code. It specifies that the symbiont must send each record in a separate I/O request to the output device.

(Valid for SJC$_ALTER_QUEUE, SJC$_CREATE_QUEUE, SJC$_START_QUEUE function codes)

SJC$_RELATIVE_PAGE

The SJC$_RELATIVE_PAGE item code is an input value item code. It is meaningful only for output execution queues. The buffer must specify a signed longword integer. This item code specifies that printing should be resumed after spacing forward (if the buffer value is positive) or backward (if the buffer value is negative) the specified number of pages.

(Valid for SJC$_START_QUEUE function code)

SJC$_REQUEUE

The SJC$_REQUEUE item code is a Boolean item code. It specifies that a job is to be requeued. By default, the job is deleted.

(Valid for SJC$_ABORT_JOB function code)

SJC$_RESTART
SJC$_NO_RESTART

The SJC$_RESTART item code is a Boolean item code. It specifies that a job can restart after a system failure or can be requeued during execution. It is the default for print jobs.

The SJC$_NO_RESTART item code is a Boolean item code. It specifies that a job cannot restart after a system failure or after a requeue operation. It is the default for batch jobs.

(Valid for SJC$_ALTER_JOB, SJC$_CREATE_JOB, SJC$_ENTER_FILE function codes)

SJC$_RETAIN_ALL_JOBS
SJC$_RETAIN_ERROR_JOBS
SJC$_NO_RETAIN_JOBS

The SJC$_RETAIN_ALL_JOBS item code is a Boolean item code. It specifies that jobs are to be retained in the queue with a completion status after they have been executed.

The SJC$_RETAIN_ERROR_JOBS item code is a Boolean item code. It specifies that jobs are to be retained only if the job completed unsuccessfully (the job's completion status has the low bit clear).

The SJC$_NO_RETAIN_JOBS item code is a Boolean item code. It specifies that jobs are not to be retained in the queue after they have completed. It is the default.

(Valid for SJC$_ALTER_QUEUE, SJC$_CREATE_QUEUE, SJC$_START_QUEUE function codes)

SJC$_SCSNODE_NAME

The SJC$_SCSNODE_NAME item code is an input value item code. It is meaningful only for execution queues in a VAXcluster environment. It specifies the name of the VAX node on which the queue is to execute. The buffer must specify a string, from 1 to 6 characters, that matches the value of the SYSGEN parameter SCSNODE in effect on the target node.

By default, the queue executes on the VAX node from which the queue is first started. For an output execution queue, you use the SJC$_DEVICE_NAME item code to specify the name of the device managed by the queue.

(Valid for SJC$_CREATE_QUEUE, SJC$_START_QUEUE function codes)

SJC$_SEARCH_STRING

The SJC$_SEARCH_STRING item code is an input value item code. It is meaningful only for output execution queues. The buffer must specify a string of no more than 63 characters. This item code specifies that printing is to resume at the page containing the first occurrence of the specified string. The search for the string proceeds in the forward direction.

(Valid for SJC$_START_QUEUE function code)

SJC$_SERVER

The SJC$_SERVER item code is a Boolean item code. It is meaningful only for output queues. It specifies that the queue being created is a server queue. The term server indicates that a user-modified or user-written symbiont process is controlling an output execution queue, or a generic queue has server execution queues as its targets.

The SJC$_BATCH, SJC$_PRINTER, SJC$_SERVER, and SJC$_TERMINAL item codes are mutually exclusive. If none of these item codes are specified, the default is SJC$_PRINTER.

(Valid for SJC$_CREATE_QUEUE function code)

SJC$_SWAP
SJC$_NO_SWAP

The SJC$_SWAP item code is a Boolean item code. It is meaningful only for batch execution queues. It specifies that jobs initiated from a queue can be swapped. It is the default.

The SJC$_NO_SWAP item code is a Boolean item code. It specifies that jobs in this queue cannot be swapped.

(Valid for SJC$_ALTER_QUEUE, SJC$_CREATE_QUEUE, SJC$_START_QUEUE function codes)

SJC$_TERMINAL
SJC$_NO_TERMINAL

The SJC$_TERMINAL item code is a Boolean item code. It is meaningful only for output queues. It specifies that the queue being created is a terminal queue.

The SJC$_BATCH, SJC$_PRINTER, SJC$_SERVER, and SJC$_TERMINAL item codes are mutually exclusive. If none of these item codes are specified, the default is SJC$_PRINTER.

The SJC$_NO_TERMINAL item code is a Boolean item code. It designates the queue type as printer rather than terminal. It is the default.

For the SJC$_START_QUEUE function code, SJC$_TERMINAL and SJC$_NO_TERMINAL are supported now for compatibility with VAX/VMS Version 4.n, but may not be supported in the future. For SJC$_CREATE_QUEUE, SJC$_NO_TERMINAL is supported for compatibility with VAX/VMS Version 4.n, and may not be supported in the future.

(Valid for SJC$_CREATE_QUEUE, SJC$_START_QUEUE function codes)

SJC$_TOP_OF_FILE

The SJC$_TOP_OF_FILE item code is a Boolean item code. It is meaningful only for output queues. It specifies that printing is to be resumed at the beginning of the file.

(Valid for SJC$_START_QUEUE function code)

SJC$_UIC

The SJC$_UIC item code is an input value item code. This value specifies the 4-byte UIC of the user on behalf of whom the request is made. By default, the UIC is taken from the requesting process.

(Valid for SJC$_CREATE_JOB, SJC$_ENTER_FILE function codes)

SJC$_USERNAME

The SJC$_USERNAME item code is an input value item code. It specifies the user name of the user on behalf of whom the request is made. The buffer must specify a string from 1 to 12 characters. By default, the user name is taken from the requesting process.

You need CMKRNL privilege to use this item code.

(Valid for SJC$_CREATE_JOB, SJC$_ENTER_FILE function codes)

SJC$_WSDEFAULT
SJC$_NO_WSDEFAULT

The SJC$_WSDEFAULT item code is an input value item code. It is meaningful only for batch jobs and execution queues. It specifies the default working set size for batch jobs or jobs initiated from a batch queue, or the default working set size of a symbiont process connected to an output queue. A symbiont process can control several output queues; however, the default

working set size of the symbiont process is established by the first queue to which it is connected. The buffer must contain a longword integer value in the range 1 through 65,535.

The SJC$_NO_WSDEFAULT item code is a Boolean item code. It specifies that the system is to determine the working set default. It is the default.

For batch jobs, the default working set size, working set quota, and working set extent (maximum size) are included in each user record in the system user authorization file (UAF). You can specify values for these items for individual jobs or for all jobs in a given queue, or for both. Table SYS–8 shows the action taken when you specify a value for SJC$_WSDEFAULT.

Table SYS–8 Working Set Decision Table

Value Specified for Job?	Value Specified for Queue?	Action Taken
No	No	Use UAF value
No	Yes	Use value for queue
Yes	Yes	Use lower of the two
Yes	No	Compare specified value with UAF value; use lower

(Valid for SJC$_ALTER_JOB, SJC$_ALTER_QUEUE, SJC$_CREATE_JOB, SJC$_CREATE_QUEUE, SJC$_ENTER_FILE, SJC$_START_QUEUE function codes)

SJC$_WSEXTENT
SJC$_NO_WSEXTENT

The SJC$_WSEXTENT item code is an input value item code. It is meaningful only for batch jobs and execution queues. It specifies the working set extent for batch jobs or jobs initiated from a batch queue, or the working set extent of a symbiont process connected to an output queue. A symbiont process can control several output queues; however, the working set extent of the symbiont process is established by the first queue to which it is connected. The buffer must contain a longword integer value in the range 1 through 65,535.

The SJC$_NO_WSEXTENT item code is a Boolean item code. It specifies that the system determine the working set extent. It is the default.

For information about the action taken when you specify a value for SJC$_WSEXTENT for a batch job or batch queue, refer to the description of the SJC$_WSDEFAULT item code and to Table SYS–8.

(Valid for SJC$_ALTER_JOB, SJC$_ALTER_QUEUE, SJC$_CREATE_JOB, SJC$_CREATE_QUEUE, SJC$_ENTER_FILE, SJC$_START_QUEUE function codes)

SJC$_WSQUOTA
SJC$_NO_WSQUOTA

The SJC$_WSQUOTA item code is an input value item code. It is meaningful only for batch jobs and execution queues. It specifies the working set quota for batch jobs or default WSQUOTA for jobs initiated from a batch queue, or the working set quota of a symbiont process connected to an output queue. A symbiont process can control several output queues; however, the working

set quota of the symbiont process is established by the first queue to which it is connected. The buffer must contain a longword integer value in the range 1 through 65,535.

The SJC$_NO_WSQUOTA item code is a Boolean item code. It specifies that the system is to determine the working set quota. It is the default.

For information about the action taken when you specify a value for SJC$_WSQUOTA for a batch job or batch queue, refer to the description of the SJC$_WSDEFAULT item code and to Table SYS–8.

(Valid for SJC$_ALTER_JOB, SJC$_ALTER_QUEUE, SJC$_CREATE_JOB, SJC$_CREATE_QUEUE, SJC$_ENTER_FILE, SJC$_START_QUEUE function codes)

iosb

VMS usage: **io_status_block**
type: **quadword (unsigned)**
access: **write only**
mechanism: **by reference**

I/O status block into which $SNDJBC writes the completion status after the requested operation has completed. The **iosb** argument is the address of the I/O status block.

At request initiation, $SNDJBC sets the value of the quadword I/O status block to 0. When the requested operation completes, $SNDJBC writes a condition value in the first longword of the I/O status block. It writes the value 0 into the second longword; this longword is unused and reserved for future use.

The condition values returned by $SNDJBC in the I/O status block are usually condition values from the JBC facility. These condition values are defined by the $JBCMSGDEF macro. In some cases, the condition value returned by $SNDJBC may be an error return from a system service or an RMS service that is used in executing the request. For the SJC$_SYNCHRONIZE_JOB request, the condition value returned is the completion status of the requested job.

The condition values returned from the JBC facility are listed under the heading CONDITION VALUES RETURNED IN THE I/O STATUS BLOCK.

Though this argument is optional, DIGITAL strongly recommends that you specify it, for the following reasons:

- If you are using an event flag to signal the completion of the service, you can test the I/O status block for a condition value to be sure that the event flag was not set by an event other than service completion.
- If you are using the $SYNCH service to synchronize completion of the service, the I/O status block is a required argument for $SYNCH.
- The condition value returned in R0 and the condition value returned in the I/O status block provide information about different aspects of the call to the $SNDJBC service. The condition value returned in R0 gives you information about the success or failure of the service call itself; the condition value returned in the I/O status block gives you information about the success or failure of the service operation. Therefore, to accurately assess the success or failure of the call to $SNDJBC, you must check the condition values returned in both R0 and the I/O status block.

astadr

VMS usage: **ast_procedure**
type: **procedure entry mask**
access: **call without stack unwinding**
mechanism: **by reference**

AST service routine to be executed when $SNDJBC completes. The **astadr** argument is the address of the entry mask of this routine.

If specified, the AST routine executes at the same access mode as the caller of $SNDJBC.

astprm

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

AST parameter to be passed to the AST service routine specified by the **astadr** argument. The **astprm** argument is this longword parameter.

DESCRIPTION

Types of Queue

The VMS batch/print facility supports several types of queue, which aid in the processing of batch and print jobs. The different types of queue can be divided into three major categories according to the way the system processes the jobs assigned to the queue. The three types of queue are execution, generic, and logical. Execution queues schedule jobs for execution; generic and logical queues transfer jobs to execution queues. Within these major classifications, queue type is further defined by the kinds of job the queue can accept for processing. Some types of execution and generic queue accept batch jobs; other types accept print jobs. Logical queues are restricted to print jobs.

You create a queue by making a call to $SNDJBC specifying the SJC$_CREATE_QUEUE function code. Item codes that you optionally specify in the call determine the type of queue you create. The following list describes the various types of execution, generic, and logical queues and indicates which item codes you need to specify to create them.

1 **Execution queue.** An execution queue schedules jobs for processing. In a VAXcluster environment, jobs are processed on the node that manages the execution queue. There are two types of execution queue:

   a. **Batch execution queue.** A batch execution queue can schedule only batch jobs for execution. A batch job executes as a detached process that sequentially runs one or more command procedures; you define the list of command procedures as part of the initial job description. You create a batch execution queue by specifying the SJC$_BATCH item code in the call to the $SNDJBC service.

   b. **Output execution queue.** An output execution queue schedules print jobs for processing by an independent symbiont process associated with the queue. The job controller sends the symbiont a list of files to process; you define this list of files as part of the initial job description. As the symbiont processes each file, it produces output for the device it controls, such as a printer or terminal.

The standard print symbiont image provided by VMS is designed to print files on hardcopy devices. User-modified or user-written symbionts also can be designed for this or any other file processing activity managed by the VMS batch/print facility. The symbiont image that executes jobs from an output queue is specified by the SJC$_PROCESSOR item code. If you omit this item code, the standard VMS print symbiont image, PRTSMB, is associated with the queue.

There are three types of output execution queue:

- **Printer execution queue.** This type of queue typically uses the standard print symbiont to direct output to a line printer. You can specify a user-provided symbiont in the SJC$_PROCESSOR item code. You create a printer execution queue by specifying the SJC$_PRINTER item code when you create the output execution queue. A printer execution queue is the default type of output execution queue.
- **Terminal execution queue.** This type of queue typically uses the standard print symbiont to direct output to a terminal printer. You can specify a user-provided symbiont in the SJC$_PROCESSOR item code. You create a terminal execution queue by specifying the SJC$_TERMINAL item code when you create the output execution queue.
- **Server execution queue.** This type of queue uses the user-modified or user-written symbiont you specify in the SJC$_PROCESSOR item code to process the files that belong to jobs in the queue. You create a server execution queue by specifying the SJC$_SERVER item code when you create the output execution queue.

When you create an output execution queue, you can initially mark it as either a printer, terminal, or server execution queue. However, when the queue is started, the symbiont process associated with the queue can change the queue type from the type designated at its creation to a printer, terminal, or server execution queue, as follows:

- When an output execution queue associated with the standard VMS print symbiont is started, the symbiont determines whether it is controlling a printer or terminal. It communicates this information to the job controller. If necessary, the job controller then changes the type designation of the output execution queue.
- When an output execution queue associated with a user-modified or user-written symbiont is started, the symbiont has the option of identifying the queue to the job controller as a server queue. If the user-written or user-modified symbiont does not notify the job controller that it wants to change the queue type designation, the output execution queue retains the queue type designation it received when it was created.

2 **Generic queue.** A generic queue holds a job until an appropriate execution queue becomes available to initiate the job; the job controller then requeues the job to the available execution queue. In a VAXcluster environment, a generic queue can direct jobs to execution queues that are located on other nodes in the VAXcluster.

You create a generic queue by specifying the SJC$_GENERIC_QUEUE item code in the call to the $SNDJBC service. You designate each execution queue to which the generic queue can direct jobs by specifying the SJC$_GENERIC_TARGET item code. Because a generic queue can direct jobs to more than one execution queue, you can specify the SJC$_GENERIC_TARGET item code up to 124 times in a single call to $SNDJBC to define a complete set of execution queues for any generic queue. If you do not specify the SJC$_GENERIC_TARGET item code, the generic queue directs jobs to any execution queue that is the same type of queue as the generic queue; that is, a generic batch queue will direct a job to any available batch execution queue, and so on. There is one exception—a generic queue will not direct work to any execution queue that was created in a call to $SNDJBC that specified the SJC$_NO_GENERIC_SELECTION item code.

There are two types of generic queue:

a. **Generic batch queue.** A generic batch queue can direct jobs only to batch execution queues. You create a generic batch queue by specifying both the SJC$_GENERIC_QUEUE and SJC$_BATCH item codes in the call to the $SNDJBC service.

b. **Generic output queue.** A generic output queue can direct jobs to any of the three types of output execution queue: printer, terminal, or server. Creating a generic output queue that directs jobs to any combination of the three types of output execution queue is possible. Typically, however, when you create a generic output queue, you specify a list of type-specific target queues. This way, the generic output queue directs jobs to a single type of output execution queue. Thus, you can control whether the jobs submitted to the generic output execution queue are output on a line printer or a terminal printer, or are sent to a server symbiont for processing. You create a generic output queue by specifying the SJC$_GENERIC_QUEUE item code in the call to the $SNDJBC service.

3 **Logical queue.** A logical queue performs the same function as a generic output queue, except that a logical queue can direct jobs to only a single printer, terminal, or server execution queue. A logical queue is only an output queue that has been assigned to transfer its jobs to one execution queue.

To change an output queue into a logical queue, you make a call to the $SNDJBC service while the output queue is in a stopped state. The call must specify the SJC$_ASSIGN_QUEUE function code and the SJC$_DESTINATION_QUEUE item code.You use the SJC$_DESTINATION_QUEUE item code to specify the execution queue to which the logical queue should direct jobs. When the logical queue is started, it automatically requeues its jobs to the specified execution queue as that execution queue becomes available. You can change a logical queue back to its original output queue definition by specifying the SJC$_DEASSIGN_QUEUE function code in a subsequent call to the $SNDJBC service.

Queue Protection

This section describes UIC-based protection checking that is performed by the $SNDJBC service to control access to queues. As an alternative to this form of protection checking, you can associate ACLs with queues using the appropriate security services. For example, the $CHANGE_ACL service allows you to create or modify ACL identifiers and their protection masks. For a complete discussion of access control lists, see the chapter on security services in the *Introduction to VMS System Services*.

There are two aspects to UIC-based queue protection:

- The queue has an associated UIC. When you create a queue, you assign it a UIC by using the SJC$_OWNER_UIC item code. If you do not specify this item code, the queue is given the default UIC [1,4].
- You may assign a queue a protection mask by specifying the SJC$_PROTECTION item code. This protection mask specifies read, write, execute, and delete access for the four categories of user: owner, group, world, and system.

In addition, certain queue operations require the caller of $SNDJBC to have certain privileges. The function codes that require privileges are listed under the heading Privileges and Restrictions.

When a job is submitted to a queue, it is assigned a UIC that is the same as the UIC of the process submitting the job, unless the SJC$_UIC item code is specified to supply a different UIC.

For each requested operation, the $SNDJBC service checks the UIC and privileges of the requesting process against the UIC of the queue, protection specified for the queue, and the privilege or privileges, if any, required for the operation. This checking is performed in a way similar to the way that the file system checks access to a file by comparing the owner UIC and protection of the file with the UIC and privileges of the requester.

Operations that apply to jobs are checked against (1) the R (read) and D (delete) protection specified for the queue in which the job is entered and (2) the owner UIC of the job. In general, R access to a job allows you to determine that the job exists; D access to a job allows you to affect the job.

Operations that apply to queues are checked against (1) the W (Write) and E (Execute) protection specified for the queue and (2) the owner UIC of the queue. In general, W access to a queue allows you to submit jobs to the queue; E access to a queue allows you to act as an operator for the queue, including the ability to affect jobs in the queue, to affect accounting, and to alter queues. OPER privilege grants E access to all queues.

Privileges and Restrictions

To specify the following function codes, the caller must have both OPER and SYSNAM privilege:

SJC$_START_QUEUE_MANAGER
SJC$_STOP_QUEUE_MANAGER

To specify the following function codes, the caller must have OPER privilege:

SJC$_CREATE_QUEUE
SJC$_DEFINE_CHARACTERISTIC
SJC$_DEFINE_FORM

SJC$_DELETE_CHARACTERISTIC
SJC$_DELETE_FORM
SJC$_DELETE_QUEUE
SJC$_START_ACCOUNTING
SJC$_STOP_ACCOUNTING

To specify the following function code, the caller must have (1) OPER privilege, (2) E access to the queue containing the specified job, or (3) R access to the specified job:

SJC$_SYNCHRONIZE_JOB

To specify the following function codes, the caller must have (1) OPER privilege, (2) E access to the specified queue, or (3) W access to the specified queue:

SJC$_ADD_FILE
SJC$_CLOSE_DELETE
SJC$_CLOSE_JOB
SJC$_CREATE_JOB
SJC$_ENTER_FILE

To specify the following function codes, the caller must have OPER privilege or E access to the specified queues or queues:

SJC$_ALTER_QUEUE
SJC$_ASSIGN_QUEUE
SJC$_DEASSIGN_QUEUE
SJC$_MERGE_QUEUE
SJC$_PAUSE_QUEUE
SJC$_RESET_QUEUE
SJC$_START_QUEUE
SJC$_STOP_QUEUE

To specify the following function codes, the caller must have (1) OPER privilege, (2) E access to the queue containing the specified job, or (3) D access to the specified job:

SJC$_ABORT_JOB
SJC$_ALTER_JOB
SJC$_DELETE_JOB

To specify the following function codes, no privilege is required:

SJC$_BATCH_CHECKPOINT
SJC$_WRITE_ACCOUNTING

To specify a base priority (using the SJC$_BASE_PRIORITY item code) higher than the base priority of the requesting process, the caller needs OPER or ALTPRI privilege.

To specify a scheduling priority (using the SJC$_PRIORITY item code) higher than the value of the SYSGEN parameter MAXQUEPRI, the caller needs OPER or ALTPRI privilege.

To specify the following item codes, the caller must have OPER privilege:

SJC$_PROTECTION
SJC$_OWNER_UIC

To specify the following item codes, the caller must have CMKRNL privilege:

SJC$_ACCOUNT_NAME
SJC$_UIC
SJC$_USERNAME

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The item list or input buffer cannot be read by the caller; or the return length buffer, output buffer, or status block cannot be written by the caller.
SS$_BADPARAM	The function code is invalid; the item list contains an invalid item code; a buffer descriptor has an invalid length; or the reserved parameter has a nonzero value.
SS$_DEVOFFLINE	The job controller process is not running.
SS$_EXASTLM	You specified the **astadr** argument, and the process has exceeded its ASTLM quota.
SS$_ILLEFC	The **efn** argument specifies an illegal event flag number.
SS$_INSFMEM	Insufficient space exists for completing the request.
SS$_MBFULL	The job controller mailbox is full.
SS$_MBTOOSML	The mailbox message is too large for the job controller mailbox.
SS$_UNASEFC	The **efn** argument specifies an unassociated event flag cluster.

CONDITION VALUES RETURNED IN THE I/O STATUS BLOCK

JBC$_NORMAL	The service completed successfully.
JBC$_DELACCESS	The file protection of the specified file, which was entered with the delete option, does not allow delete access to the caller.
JBC$_DUPFORM	The specified form number is invalid because it is already defined; each form must have a unique form number.
JBC$_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$_INCFORMPAR	The specified length, width, and margin parameters are inconsistent; the value of the difference between the top and bottom margin parameters must be less than the form length, and the difference between the left and right margin parameters must be less than the line width.

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$_INVCHANAM	A specified characteristic name is not syntactically valid.
JBC$_INVDSTQUE	The destination queue name is not syntactically valid.
JBC$_INVFORNAM	The form name is not syntactically valid.
JBC$_INVFUNCOD	The specified function code is invalid.
JBC$_INVITMCOD	The item list contains an invalid item code.
JBC$_INVPARLEN	The length of a specified string is outside the valid range for that item code.
JBC$_INVPARVAL	A parameter value specified for an item code is outside the valid range for that item code.
JBC$_INVQUENAM	The queue name is not syntactically valid.
JBC$_JOBQUEDIS	The request cannot be executed because the system job queue manager has not been started.
JBC$_JOBQUEENA	The system job queue manager cannot be started because it is already running.
JBC$_MISREQPAR	An item code that is required for the specified function code has not been specified.
JBC$_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$_NORESTART	The specified job cannot be requeued because it was not defined to be restartable.
JBC$_NOSUCHCHAR	The specified characteristic does not exist.

JBC$_NOSUCHFORM	The specified form does not exist.
JBC$_NOSUCHJOB	The specified job does not exist.
JBC$_NOSUCHNODE	The specified node does not exist.
JBC$_NOSUCHQUE	The specified queue does not exist.
JBC$_NOTASSIGN	The specified queue cannot be deassigned because it is not assigned.
JBC$_QUENOTSTOP	The specified queue cannot be deleted because it is not in a stopped state.
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

The following FORTRAN program demonstrates the use of the $SNDJBCW service to submit a batch job that is to execute on behalf of another user. No log file is produced for the batch job. This program saves the job's entry number. You need CMKRNL privilege to run this program.

```
! Declare system service related symbols
INTEGER*4       SYS$SNDJBCW,
2               STATUS
INCLUDE         '($SJCDEF)'

! Define item list structure
STRUCTURE       /ITMLST/
  UNION
    MAP
      INTEGER*2 BUFLEN, ITMCOD
      INTEGER*4 BUFADR, RETADR
    END MAP
    MAP
      INTEGER*4 END_LIST
    END MAP
  END UNION
END STRUCTURE

! Define I/O status block structure
STRUCTURE       /IOSBLK/
INTEGER*4       STS, ZEROED
END STRUCTURE

! Declare $SNDJBCW item list and I/O status block
RECORD /ITMLST/ SUBMIT_LIST(6)
RECORD /IOSBLK/ IOSB

! Declare variables used in $SNDJBCW item list
CHARACTER*9     QUEUE                   /'SYS$BATCH'/
CHARACTER*23    FILE_SPECIFICATION      /'$DISK1:[COMMON]TEST.COM'/
CHARACTER*12    USERNAME                /'PROJ3036    '/
INTEGER*4       ENTRY_NUMBER
```

```
! Initialize item list for the enter file operation
SUBMIT_LIST(1).BUFLEN = 9
SUBMIT_LIST(1).ITMCOD = SJC$_QUEUE
SUBMIT_LIST(1).BUFADR = %LOC(QUEUE)
SUBMIT_LIST(1).RETADR = 0
SUBMIT_LIST(2).BUFLEN = 23
SUBMIT_LIST(2).ITMCOD = SJC$_FILE_SPECIFICATION
SUBMIT_LIST(2).BUFADR = %LOC(FILE_SPECIFICATION)
SUBMIT_LIST(2).RETADR = 0
SUBMIT_LIST(3).BUFLEN = 12
SUBMIT_LIST(3).ITMCOD = SJC$_USERNAME
SUBMIT_LIST(3).BUFADR = %LOC(USERNAME)
SUBMIT_LIST(3).RETADR = 0
SUBMIT_LIST(4).BUFLEN = 0
SUBMIT_LIST(4).ITMCOD = SJC$_NO_LOG_SPECIFICATION
SUBMIT_LIST(4).BUFADR = 0
SUBMIT_LIST(4).RETADR = 0
SUBMIT_LIST(5).BUFLEN = 4
SUBMIT_LIST(5).ITMCOD = SJC$_ENTRY_NUMBER_OUTPUT
SUBMIT_LIST(5).BUFADR = %LOC(ENTRY_NUMBER)
SUBMIT_LIST(5).RETADR = 0
SUBMIT_LIST(6).END_LIST = 0

! Call $SNDJBCW service to submit the batch job
STATUS = SYS$SNDJBCW (,
2               %VAL(SJC$_ENTER_FILE),,
2               SUBMIT_LIST,
2               IOSB,,)
IF (STATUS) STATUS = IOSB.STS
IF (.NOT. STATUS) CALL LIB$SIGNAL (%VAL(STATUS))
END
```

$SNDJBCW Send to Job Controller and Wait for Completion

The Send to Job Controller and Wait for Completion and $GETQUI services together provide the user interface to the Job Controller (JBC) facility. The $SNDJBW service allows you to create, stop, and manage queues and the jobs in those queues. Queues may be generic, batch, execution, or output queues. Jobs may be batch or print jobs.

The $SNDJBCW service queues a request to the Job Controller. For most operations, $SNDJBCW completes synchronously; that is, it returns to the caller after the operation completes. However, if the requested operation is a pause queue, stop queue, or abort job operation, $SNDJBCW returns to the caller after queuing the request. There is no way to synchronize completion of these operations. Also, $SNDJBCW does not wait for a job to complete before it returns to the caller; to synchronize completion of a job, the caller must specify the SJC$_SYNCHRONIZE_JOB function code.

The $SNDJBCW service is identical to the Send to Job Controller ($SNDJBC) service except that $SNDJBC completes asynchronously; the $SNDJBC service returns to the caller immediately after queuing the request, without waiting for the operation to complete.

For additional information about the $SNDJBCW service, refer to the documentation of the $SNDJBC service.

The $SNDJBC and $SNDJBCW services supersede the Send Message to Symbiont Manager ($SNDSMB) and Send Message to Accounting Manager ($SNDACC) services. You should write new programs using $SNDJBC or $SNDJBCW, instead of $SNDSMB or $SNDACC. You should convert old programs using $SNDSMB or $SNDACC to use $SNDJBC or $SNDJBCW, as convenient.

FORMAT

SYS$SNDJBCW *[efn],func[,nullarg][,itmlst][,iosb][,astadr][,astprm]*

$SNDOPR Send Message to Operator

The $SNDOPR service performs the following functions:

- Sends a user request to operator terminals
- Sends a user cancellation request to operator terminals
- Sends an operator reply to a user terminal
- Enables an operator terminal
- Displays the status of an operator terminal
- Initializes the operator log file

FORMAT

SYS$SNDOPR *msgbuf ,[chan]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns 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**

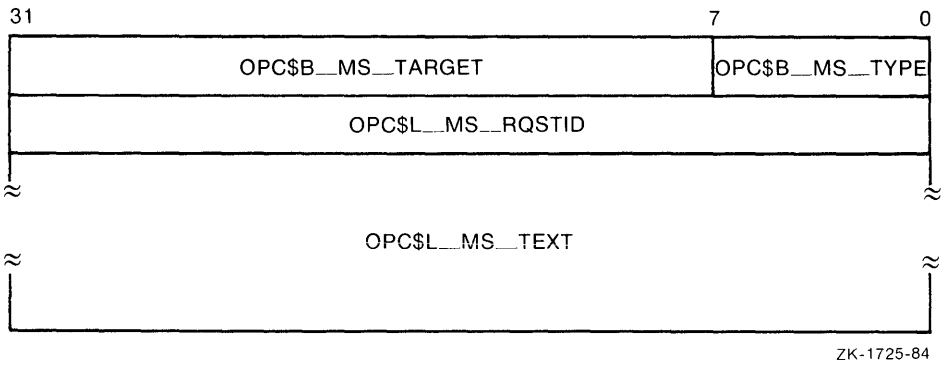
User buffer specifying the operation to be performed and the information needed to perform that operation. The **msgbuf** argument is the address of a character string descriptor pointing to the buffer.

The format and contents of the buffer vary with the requested operation; however, the first byte in any buffer is the request code, which specifies the operation to be performed. The $OPCMSG macro defines the symbolic names for these request codes. The following table shows each operation that $SNDOPR performs and the request code that specifies that operation.

Request Code	Corresponding Operation
OPC$_RQ_RQST	Sends a user request to operator terminals. This request code is used to make an operator request. If you specify a reply to the request (by using the **chan** argument), the operator request is kept active until the operator responds.
OPC$_RQ_CANCEL	Sends a user cancellation request to specified operator terminals. You use this request code to notify one or more operators that a previous request is to be cancelled. To specify OPC$_RQ_CANCEL, you must also specify the **chan** argument.
OPC$_RQ_REPLY	Sends an operator reply to a user who has made a request. Operators use this request code to report the status of a user request. The format of the message buffer for this request is the format of the reply found in the user's mailbox after the call to $SNDOPR completes. All functions of $SNDOPR that deliver a reply to a mailbox do so in the format described for this request code.
OPC$_RQ_TERME	Enables an operator terminal. You use this request to enable a specified terminal to receive operator messages.
OPC$_RQ_STATUS	Reports the status of an operator terminal. Operators use this request to display the operator classes for which the specified terminal is enabled and a list of outstanding requests.
OPC$_RQ_LOGI	Initializes the operator log file.

The following diagrams depict the message buffer for each of these request codes. Each field within a diagram has a symbolic name, which serves to identify the field; in other words, these names specify offsets into the message buffer. The list following each diagram shows each field name and what its contents can or should be. The $OPCDEF macro defines the field names, as well as any other symbolic name that may be specified as the contents of a field.

Message Buffer Format for OPC$_RQ_RQST



OPC$B_MS_TYPE — This 1-byte field contains the request code OPC$_RQ_RQST.

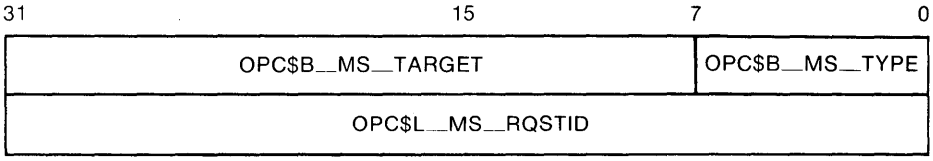
OPC$B_MS_TARGET — This 3-byte field contains a 24-bit bit vector that specifies which operator terminal types are to receive the request. The $OPCDEF macro defines symbolic names for the operator terminal types. You construct the bit vector by specifying the desired symbolic names in a logical OR operation. Following is the symbolic name of each operator terminal type:

OPC$M_NM_CARDS	Card device operator
OPC$M_NM_CENTRL	Central operator
OPC$M_NM_CLUSTER	VAXcluster operator
OPC$M_NM_DEVICE	Device status information
OPC$M_NM_DISKS	Disk operator
OPC$M_NM_NTWORK	Network operator
OPC$M_NM_TAPES	Tape operator
OPC$M_NM_PRINT	Printer operator
OPC$M_NM_SECURITY	Security operator
OPC$M_NM_OPER1	OPC$M_NM_OPER1 through
to	OPC$M_NM_OPER12 specify
OPC$M_NM_OPER12	system manager-defined operator functions.

OPC$L_MS_RQSTID — This longword field contains a user-supplied longword message code.

OPC$L_MS_TEXT — This variable-length field contains an ASCII string specifying text to be sent to the specified operator terminals. The length of the string must be in the range 0 to 255 bytes.

Message Buffer Format for OPC$_RQ_CANCEL



ZK-1726-84

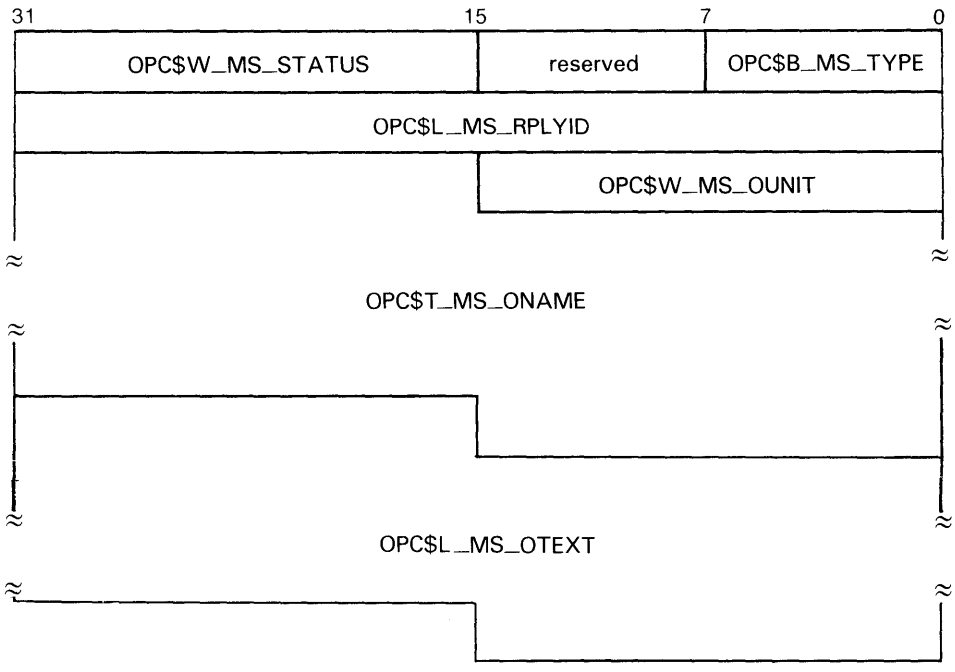
OPC$B_MS_TYPE — This 1-byte field contains the request code OPC$_RQ_CANCEL.

OPC$B_MS_TARGET — This 3-byte field contains a 24-bit bit vector that specifies which operator terminal types are to receive the cancellation request. The $OPCDEF macro defines symbolic names for the operator terminal types. You construct the bit vector by specifying the desired symbolic names in a logical OR operation. Following is the symbolic name of each operator terminal type:

OPC$M_NM_CARDS	Card device operator
OPC$M_NM_CENTRL	Central operator
OPC$M_NM_SECURITY	Security operator
OPC$M_NM_CLUSTER	VAXcluster operator
OPC$M_NM_DEVICE	Device status information
OPC$M_NM_DISKS	Disk operator
OPC$M_NM_NTWORK	Network operator
OPC$M_NM_TAPES	Tape operator
OPC$M_NM_PRINT	Printer operator
OPC$M_NM_OPER1	OPC$M_NM_OPER1 through
to	OPC$M_NM_OPER12 specify
OPC$M_NM_OPER12	system manager-defined operator functions.

OPC$L_MS_RQSTID — This longword field contains a user-supplied longword message code.

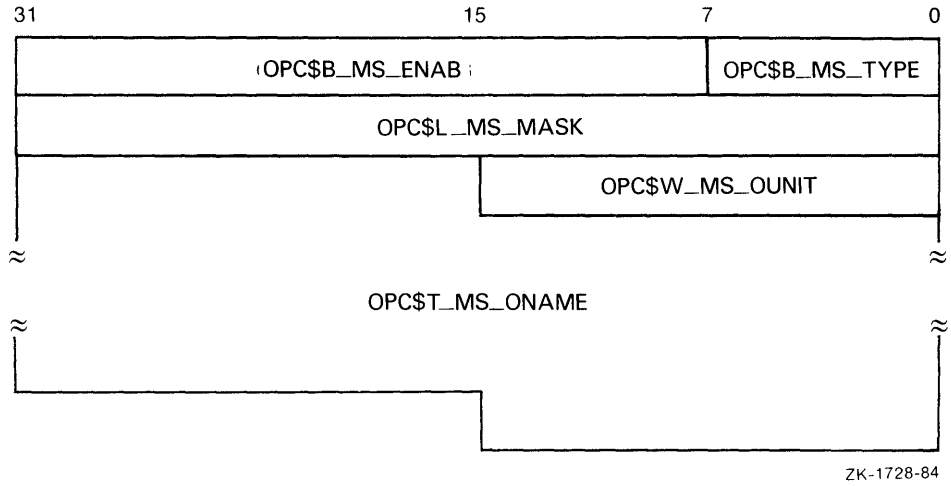
Message Buffer Format for OPC$_RQ_REPLY



ZK-1727-84

OPC$B_MS_TYPE	This 1-byte field contains the request code OPC$_RQ_REPLY.
Reserved	This 1-byte field is reserved for future use.
OPC$W_MS_STATUS	This 2-byte field contains the low-order word of the longword condition value that $SNDOPR returns in the mailbox specified by the **chan** argument. You can find a list of these longword condition values under CONDITION VALUES RETURNED IN THE MAILBOX. To test the completion status, you need to extract the low-order word from the longword condition value and compare it to the contents of the OPC$W_MS_STATUS field.
OPC$L_MS_RPLYID	This 4-byte field contains a user-supplied message code.
OPC$W_MS_OUNIT	This 2-byte field contains the unit number of the terminal to which the operator reply is to be sent. To obtain the unit number of the terminal, you can call $GETDVI, specifying the DVI$_FULLDEVNAM item code. The information returned will consist of the node name and device name as a padded string. Because the unit number is found on the tail end of the string, you must parse the string. One way to do this is, starting from the tail end, to search for the first alphabetic character; the digits to the right of this alphabetic character constitute the unit number. After extracting the unit number, count the remaining characters in the string. Then, construct a counted ASCII string and use this for the following field, OPC$T_MS_ONAME.
OPC$T_MS_ONAME	This variable-length field contains a counted ASCII string specifying the device name of the terminal that is to receive the operator reply. The maximum total length of the string is 14 bytes. See the preceding field description (OPC$T_MS_OUNIT) to learn how to obtain the device name.
OPC$L_MS_OTEXT	This variable-length field contains an ASCII string specifying operator-written text to be sent to the user terminal. The length of the string must be in the range 0 to 255 bytes. This field is optional.

Message Buffer Format for OPC$_RQ_TERME



OPC$B_MS_TYPE — This 1-byte field contains the request code OPC$_RQ_TERME.

OPC$B_MS_ENAB — This 3-byte field contains a user-supplied value. The value 0 indicates that the specified terminal is to be disabled for the specified operator classes. Any nonzero value indicates that the specified terminal is to be enabled for the specified operator classes.

OPC$B_MS_MASK

This 4-byte field contains a 4-byte bit vector that specifies which operator terminal types are to be enabled or disabled for the specified terminal. The $OPCDEF macro defines symbolic names for the operator terminal types. You construct the bit vector by specifying the desired symbolic names in a logical OR operation. Following is the symbolic name of each operator terminal type:

OPC$M_NM_CARDS	Card device operator
OPC$M_NM_CENTRL	Central operator
OPC$M_NM_SECURITY	Security operator
OPC$M_NM_CLUSTER	VAXcluster operator
OPC$M_NM_DEVICE	Device status information
OPC$M_NM_DISKS	Disk operator
OPC$M_NM_NTWORK	Network operator
OPC$M_NM_TAPES	Tape operator
OPC$M_NM_PRINT	Printer operator
OPC$M_NM_OPER1	OPC$M_NM_OPER1 through
to	OPC$M_NM_OPER12 specify
OPC$M_NM_OPER12	system manager-defined operator functions.

OPC$W_MS_OUNIT

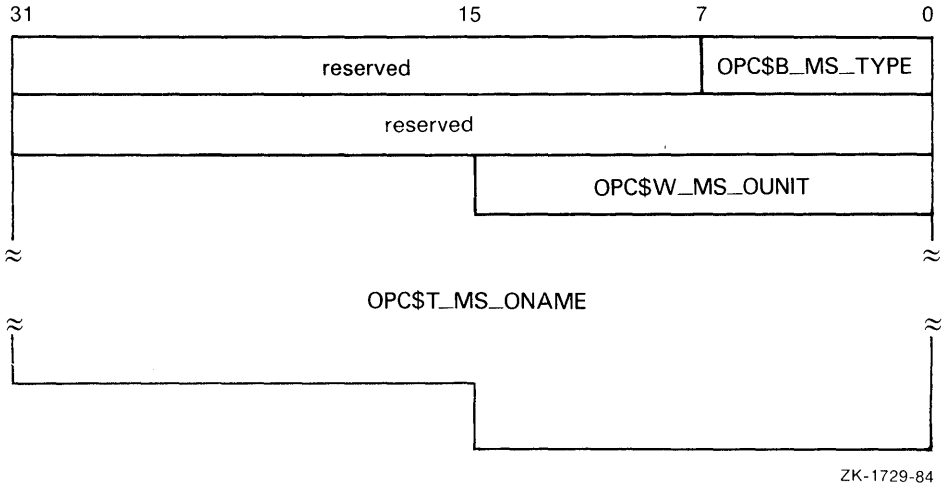
This 2-byte field contains the unit number of the operator terminal to be enabled or disabled for the specified operator terminal types. To obtain the unit number of the terminal, you can call $GETDVI, specifying the DVI$_FULLDEVNAM item code. The information returned will consist of the node name and device name as a padded string. Because the unit number is found on the tail end of the string, you must parse the string. One way to do this is, starting from the tail end, to search for the first alphabetic character; the digits to the right of this alphabetic character constitute the unit number.

After extracting the unit number, count the remaining characters in the string. Then, construct a counted ASCII string and use this for the following field, OPC$T_MS_ONAME.

OPC$T_MS_ONAME

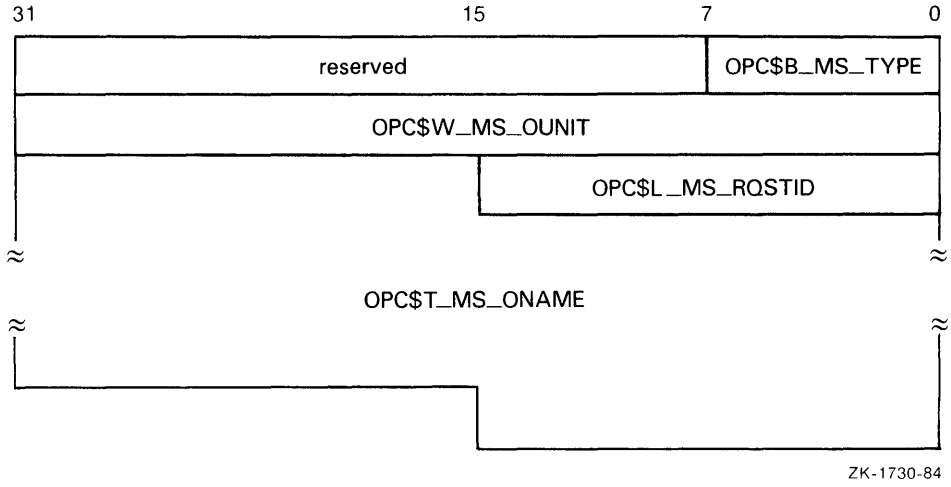
This variable-length field contains a counted ASCII string specifying the device name of the operator terminal to be enabled or disabled for the specified operator terminal types. The maximum total length of the string is 16 bytes. See the preceding field description (OPC$T_MS_OUNIT) to learn how to obtain the device name.

Message Buffer Format for OPC$_RQ_STATUS



OPC$B_MS_TYPE	This 1-byte field contains the request code OPC$_RQ_STATUS.
Reserved	This 3-byte field is reserved for future use.
Reserved	This 4-byte field is reserved for future use.
OPC$W_MS_OUNIT	This 2-byte field contains the unit number of the operator terminal whose status is to be requested. To obtain the unit number of the terminal, you can call $GETDVI, specifying the DVI$_FULLDEVNAM item code. The information returned will consist of the node name and device name as a padded string. Because the unit number is found on the tail end of the string, you must parse the string. One way to do this is, starting from the tail end, to search for the first alphabetic character; the digits to the right of this alphabetic character constitute the unit number. After extracting the unit number, count the remaining characters in the string. Then, construct a counted ASCII string and use this for the following field, OPC$T_MS_ONAME.
OPC$T_MS_ONAME	This variable-length field contains a counted ASCII string specifying the device name of the operator terminal whose status is requested. The maximum total length of the string is 14 bytes. See the preceding field description (OPC$T_MS_OUNIT) to learn how to obtain the device name.

Message Buffer Format for OPC$_RQ_LOGI



OPC$B_MS_TYPE	This 1-byte field contains the request code OPC$_RQ_LOGI.
Reserved	This 3-byte field is reserved for future use.
OPC$L_MS_RQSTID	This longword field contains a user-supplied value. The value 0 specifies that the current operator log file is to be closed and a new log file opened. The value 1 specifies that the current operator log file is to be closed but no new log file is to be opened.
OPC$W_MS_OUNIT	This 2-byte field contains the unit number of the operator terminal that is making the initialization request. To obtain the unit number of the terminal, you can call $GETDVI, specifying the DVI$_FULLDEVNAM item code. The information returned will consist of the node name and device name as a padded string. Because the unit number is found on the tail end of the string, you must parse the string. One way to do this is, starting from the tail end, to search for the first alphabetic character; the digits to the right of this alphabetic character constitute the unit number. After extracting the unit number, count the remaining characters in the string. Then, construct a counted ASCII string and use this for the following field, OPC$T_MS_ONAME.
OPC$T_MS_ONAME	This variable-length field contains a counted ASCII string specifying the device name of the operator terminal that is making the initialization request. The maximum total length of the string is 14 bytes. See the preceding field description (OPC$T_MS_OUNIT) to learn how to obtain the device name.

chan

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

Channel assigned to the mailbox to which the reply is to be sent. The **chan** argument is a longword value containing the number of the channel. If you do not specify **chan** or specify it as 0 (the default), no reply is sent.

If a reply from the operator is desired, you must specify the **chan** argument.

DESCRIPTION

Depending on the operation, the calling process may need to have OPER privilege to use $SNDOPR for the following functions:

- Enable a terminal as an operator's terminal.
- Reply to or cancel a user's request.
- Initialize the operator communication log file.

In addition, the operator must have SECURITY privilege as well as OPER privilege to affect security functions.

The Send Message To Operator system 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. Call the $SNDOPR service.
3. Check the condition value returned in R0 to ensure the request was successfully made.
4. Issue a read request to the mailbox specified, if any.
5. When the read completes, check the 2-byte condition value in the OPC$W_MS_STATUS field to ensure that the operation was performed successfully.

The format of messages displayed on operator terminals follows:

```
%%%%%%%%%%  OPCOM   dd-mmm-yyyy hh:mm:ss.cc
message specific information
```

The following example shows the message displayed on a terminal as a result of a request to enable that terminal as an operator terminal:

```
%%%%%%%%%%  OPCOM   30-DEC-1988 13:44:40.37
Operator _NODE$LTA5: has been enabled, username HOEBLE
```

The following example shows the message displayed on an operator terminal as a result of a request to display the status of that operator terminal:

```
%%%%%%%%%%  OPCOM   30-DEC-1988 12:11:10.48
Operator status for operator _NODE$OPA0:
CENTRAL, PRINTER, TAPES, DISKS, DEVICES, CARDS, CLUSTER, SECURITY,
OPER1, OPER2, OPER3, OPER4, OPER5, OPER6, OPER7, OPER8, OPER9,
OPER10, OPER11, OPER12
```

The following example shows the message displayed on an operator terminal as a result of a user request:

```
%%%%%%%%%%  OPCOM   30-DEC-1988 12:57:32.25
Request 1285, from user ROSS on NODE_NAME
Please mount device _NODE$DMA0:
```

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The message buffer or buffer descriptor cannot be read by the caller.
SS$_BADPARAM	The specified message has a length of 0 or has more than 986 bytes.
SS$_DEVNOTMBX	The channel specified is not assigned to a mailbox.
SS$_INSFMEM	The system dynamic memory is insufficient for completing the service.
SS$_IVCHAN	You specified an invalid channel number. An invalid channel number is one that is 0 or a number larger than the number of channels available.
SS$_NOPRIV	The process does not have the privilege to reply to or cancel a user's request; the process does not have read/write access to the specified mailbox; or the channel was assigned from a more privileged access mode.

CONDITION VALUES RETURNED IN THE MAILBOX

OPC$_BLANKTAPE	The service completed successfully; the operator responded with the DCL command REPLY /BLANK_TAPE=n.
OPC$_INITAPE	The service completed successfully; the operator responded with the DCL command REPLY /INITIALIZE_TAPE=n.
OPC$_NOPERATOR	The service completed successfully; no operator terminal was enabled to receive the message.
OPC$_RQSTCMPLTE	The service completed successfully; the operator completed the request.
OPC$_RQSTPEND	The service completed successfully; the operator will perform the request when possible.
OPC$_RQSTABORT	The operator could not satisfy the request.
OPC$_RQSTCAN	The caller canceled the request.

EXAMPLES

1

```
;++
; Build and send an operator request.
;--

        $dscdef                         ; Define descriptor offsets
        $opcdef                         ; Define OPCOM message offsets
                                        ; and codes
        $opcmsg                         ; Define message type codes

;
; Local storage and data
;

bufsiz  = <opc$l_ms_text+120>           ; Maximum request buffer size

rqstprmpt:                              ; Prompt for user request
        .ascid  /Request> /

rqst:                                   ; User request text
                                        ; (dynamic string)
        .word   0
        .byte   dsc$k_dtype_t
        .byte   dsc$k_class_d
        .long   0

msgdsc:                                 ; Desciptor of request
                                        ; message buffer
        .long   bufsiz
        .address msgbuf

msgbuf:                                 ; Request message buffer
        .blkb   bufsiz

rqstid:                                 ; User request ID number
        .long   0
        .page
        .sbttl  Main routine
;+
; Prompt user for request text.
;
; Build the request message.
;
; Send the request to the operator.
;-
        .entry  oprexample,^m<r2,r3,r4,r5,r6,r7,r8,r9,r10,r11>
        ;
        ; Prompt user for request text.
        ;
        movaq   rqstprmpt,r2            ; Get address of prompt string
        movaq   rqst,r3                 ; Get address of result buffer desc.
prompt: pushaq  (r2)                    ; Prompt string
        pushaq  (r3)                    ; Result buffer
        calls   #2,g^lib$get_input      ; Get the request text
        blbs    r0,10$                  ; Branch if success
        ret                             ; Return error status
10$:    tstw    dsc$w_length(r3)        ; Check for text
        beql    prompt                  ; Branch if none - try again
        ;
        ; Build the request message.
        ;
        movab   msgbuf,r4               ; Get address
        movb    #opc$_rq_rqst,-         ; Insert message type
                opc$b_ms_type(r4)       ;
```

```
insv    #opc$m_nm_disks,-       ; Insert target mask (disks)
        #0,-                    ;       starting at bit 0
        #24,-                   ;       continue for 24 bits
        opc$b_ms_target(r4)     ;       into the TARGET field
moval   rqstid,r5               ; Get address of request id
incl    (r5)                    ; Set to next request number
movl    (r5),opc$l_ms_rqstid(r4); Insert request number
pushr   #^m<r2,r3,r4,r5>        ; Save registers
movc5   dsc$w_length(r3),-      ; Copy request text
                                ;    to message buffer
        @dsc$a_pointer(r3),-    ;
        #0,-                    ; Fill with zeros
        #120,-                  ; Truncate to 120 characters
        opc$l_ms_text(r4)       ;
popr    #^m<r2,r3,r4,r5>        ; Restore registers
movaq   msgdsc,r6               ; Get address of
                                ;    message descriptor
addw3   #opc$l_ms_text,-        ; Calculate message length
        dsc$w_length(r3),-      ;
        dsc$w_length(r6)        ;
;
; Send the request to the operator.
;
$sndopr_s msgdsc                ; Send request
                                ;    (no reply expected)
ret                             ; Return to caller
.end    oprexample
```

This VAX MACRO example allows you to build an operator request and send the request to the operator.

```
2       IMPLICIT NONE

        ! Symbol definitions
        INCLUDE '($DVIDEF)'
        INCLUDE '($OPCDEF)'

        ! Structures for SNDOPR
        STRUCTURE /MESSAGE/
         UNION
          MAP
           BYTE TYPE,
        2       ENABLE(3)
           INTEGER*4 MASK
           INTEGER*2 OUNIT
           CHARACTER*14 ONAME
          END MAP
          MAP
           CHARACTER*24 STRING
          END MAP
         END UNION
        END STRUCTURE
        RECORD /MESSAGE/ MSGBUF
        ! Length of MSGBUF.ONAME
        INTEGER*4 ONAME_LEN

        ! Status and routines
        INTEGER*4 STATUS,
        2         LIB$GETDVI,
        2         SYS$SNDOPR
```

```
! Type
MSGBUF.TYPE = OPC$_RQ_TERME
! Enable
MSGBUF.ENABLE(1) = 1
! Operator type
MSGBUF.MASK = OPC$M_NM_OPER1
! Terminal unit number
STATUS = LIB$GETDVI (DVI$_UNIT,
2                    ,
2                    'SYS$OUTPUT',
2                    MSGBUF.OUNIT,,)
IF (.NOT. STATUS) CALL LIB$SIGNAL (%VAL(STATUS))
! Terminal name
STATUS = LIB$GETDVI (DVI$_FULLDEVNAM,
2                    ,
2                    'SYS$OUTPUT',,
2                    MSGBUF.ONAME,
2                    ONAME_LEN)
IF (.NOT. STATUS) CALL LIB$SIGNAL (%VAL(STATUS))
! Remove unit number from ONAME and set up counted string
ONAME_LEN = ONAME_LEN - 3
MSGBUF.ONAME(2:ONAME_LEN+1) = MSGBUF.ONAME(1:ONAME_LEN)
MSGBUF.ONAME(1:1) = CHAR(ONAME_LEN)
! Call $SNDOPR
STATUS = SYS$SNDOPR (MSGBUF.STRING,)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
END
```

This VAX FORTRAN program enables the current terminal to receive OPER1 operator messages.

$SUSPND Suspend Process

The Suspend Process service allows a process to suspend itself or another process.

A suspended process can receive executive or kernel mode ASTs, unless it is suspended at kernel mode. If a process is suspended at kernel mode, the process cannot receive any ASTs or otherwise be executed until another process resumes or deletes it.

FORMAT

SYS$SUSPND *[pidadr] ,[prcnam] ,[flags]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

pidadr

VMS usage: **process_id**
type: **longword (unsigned)**
access: **modify**
mechanism: **by reference**

Process identification (PID) of the process to be suspended. The **pidadr** argument is the address of the longword PID.

You must specify the **pidadr** argument to suspend a process whose UIC group number is different from that of the calling process.

prcnam

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

Name of the process to be suspended. The **prcnam** argument is the address of a character string descriptor pointing to a 1- to 15-character process name string.

A process name is implicitly qualified by its UIC group number. Because of this, you can use the **prcnam** argument only to suspend processes in the same UIC group as the calling process.

To suspend processes in other groups, you must specify the **pidadr** argument.

If you specify neither the **pidadr** nor **prcnam** argument, the caller process is suspended.

flags

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

Longword of bit flags specifying options for the suspend operation. Currently, only bit 0 is used for the **flags** argument. When bit 0 is set, the process should be suspended at kernel mode and ASTs are not deliverable to the process.

To request a kernel mode suspend, the caller must be in either kernel mode or executive mode. For Version 5.0 of VMS, the default (bit 0 is clear) is to suspend the process at supervisor mode, where executive or kernel mode ASTs can be delivered to the process. If executive or kernel mode ASTs have been delivered to a process suspended at supervisor mode, that process will return to its suspended state after the AST routine executes.

DESCRIPTION

Depending on the operation, the calling process may need one of the following privileges to use $SUSPND:

- GROUP privilege to suspend another process in the same group, unless the process to be suspended has the same UIC as the calling process
- WORLD privilege to suspend any other process in the system

The $SUSPND service requires system dynamic memory.

The $SUSPND service completes successfully if the target process is already suspended.

Unless it has pages locked in the balance set, a suspended process can be removed from the balance set to allow other processes to execute.

Note that a kernel-mode suspend request can override a supervisor-mode suspend state, but a supervisor suspend request cannot override a kernel-mode suspend state.

The Resume Process ($RESUME) service allows a suspended process to continue. If one or more resume requests are issued for a process that is not suspended, a subsequent suspend request completes immediately; that is, the process is not suspended. No count is maintained of outstanding resume requests.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The process name string or string descriptor cannot be read by the caller, or the process identification cannot be written by the caller.
SS$_INSFMEM	The system dynamic memory is insufficient for completing the service.
SS$_IVLOGNAM	The specified process name has a length of 0 or has more than 15 characters.

SS$_NONEXPR	The specified process does not exist, or an invalid process identification was specified.
SS$_NOPRIV	The target process was not created by the caller and the calling process does not have GROUP or WORLD privilege.

$SYNCH Synchronize

The Synchronize service checks the completion status of a system service that completes asynchronously. The service whose completion status is to be checked must have been called with the **efn** and **iosb** arguments specified, because the $SYNCH service uses the event flag and I/O status block of the service to be checked.

Refer to the *Introduction to VMS System Services* for a complete discussion of system service completion.

FORMAT

SYS$SYNCH *[efn] ,[iosb]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

efn

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

Number of the event flag specified in the call to the system service whose completion status is to be checked by $SYNCH. The **efn** argument is a longword containing this number; however, $SYNCH uses only the low-order byte.

iosb

VMS usage: **io_status_block**
type: **quadword (unsigned)**
access: **write only**
mechanism: **by reference**

I/O status block specified in the call to the system service whose completion status is to be checked by $SYNCH. The **iosb** argument is the address of this quadword I/O status block.

DESCRIPTION

The $SYNCH service performs a true test for the completion of an asynchronous service such as $GETJPI. The $SYNCH service operates in the following way:

1 When called, $SYNCH waits (by calling the $WAITFR service) for the event flag to be set.

2 When the event flag is set, $SYNCH checks to see whether the I/O status block is nonzero. If it is nonzero, then the asynchronous service has completed, and $SYNCH returns to the caller.

3 If the I/O status block is zero, then the asynchronous service has not yet completed and the event flag was set by the completion of an event not associated with the completion of $GETJPI. In this case, $SYNCH clears the event flag (by calling the $CLREF service) and waits again (by calling $WAITFR) for the event flag to be set, repeating this cycle until the I/O status block is nonzero.

The $SYNCH service always sets the specified event flag when it returns to the caller. This ensures that different program segments can use the same event flag without clashing. For example, assume that calls to $GETJPI and $GETSYI both specify the same event flag and that $SYNCH is called to check for the completion of $GETJPI. If $GETSYI sets the event flag, $SYNCH clears the flag and waits for $GETJPI to set it. When $GETJPI sets the flag, $SYNCH returns to the caller and sets the event flag. In this way, the flag set by $GETSYI is not lost, and another call to $SYNCH will show the completion of $GETSYI.

The $SYNCH service is useful when a program calls an asynchronous service but must perform some other work before testing for the completion of the asynchronous service. In this case, the program should call $SYNCH at that point when it must know that the service has completed and when it is willing to wait for the service to actually complete.

When a program calls an asynchronous service (for example, $QIO) and actually waits in line (by calling $WAITFR) for its completion without performing any other work, you could improve that program by calling the synchronous form of that service (for example, $QIOW). The synchronous services such as $QIOW execute code that checks for the true completion status in the same way that $SYNCH does.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully. The asynchronous service has completed, and the I/O status block contains the condition value describing the completion status of the asynchronous service.

SYS$RMSRUNDWN RMS Rundown

The RMS Rundown service closes all files opened by RMS for the image or process and halts I/O activity. This routine performs a $CLOSE service for each file opened for processing.

FORMAT

SYS$RMSRUNDWN *buf-addr, type-value*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

buf-addr

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

A descriptor pointing to a 22-byte buffer that is to receive the device identification (16 bytes) and the file identification (6 bytes) of an improperly closed output file. The **buf-addr** argument is the address of the descriptor that points to the buffer.

type-value

VMS usage: **byte_unsigned**
type: **byte (unsigned)**
access: **read only**
mechanism: **by value**

A single byte code that specifies the type of I/O rundown to be performed. The **type-value** argument is the actual value used.

This type of code has the following values and meanings:

0

Rundown of image and indirect I/O for process permanent files.

1

Rundown of image and process permanent files: the caller's mode must not be user.

2

Abort RMS I/O: the caller's mode must be either executive or kernel (the system calls the I/O rundown control routine with this argument for process deletion).

DESCRIPTION

In addition to closing all files and terminating I/O activity, the I/O rundown control routine releases all locks held on records in shared files, clears buffers, and returns other resources allocated for file processing. You should continue to call the rundown control routine until you receive the success completion status code of RMS$_NORMAL.

Note that, prior to the execution of the $CLOSE service, the rundown control routine cancels all outstanding file operations specified in a FAB control block or any QIO requests related to file operations (an Open, Create, or Extend service, for example). It also cancels any read/write requests to nondisk devices such as terminals or mailboxes prior to the execution of the Close service, resulting in possible loss of data. All read/write requests of disk I/O buffers, however, are allowed to complete, which guarantees that none of the data written to disk files will be lost.

There is no predefined macro of the form $RMSRUNDWN_G or $RMSRUNDWN_S to call this service.

CONDITION VALUES RETURNED

RMS$_NORMAL	The service completed successfully.
RMS$_CCF	The I/O rundown routine cannot close the file.
RMS$_IAL	The argument list is invalid. An output file could not be closed successfully, and the user buffer could not be written.

SYS$SETDDIR Set Default Directory

The Set Default Directory service allows you to read and change the default directory string for the process. You should restore the old default directory string to its original status unless you want the changed default directory string to last beyond the exit of your image.

FORMAT

SYS$SETDDIR *[new-dir-addr] [,length-addr] [,cur-dir-addr]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

new-dir-addr

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

A descriptor of the new default directory. The **new-dir-addr** argument is the address of the descriptor that points to the buffer containing the new directory specification that RMS will use to set the new process-default directory. If the default directory is not to be changed, the value of the **new-dir-addr** argument should be 0.

length-addr

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

A word that is to receive the length of the current default directory. The **length-addr** argument is the address of the word that will receive the length. If you do not want this value returned, specify the value 0.

cur-dir-addr

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

A descriptor of a buffer that is to receive the current default directory string. The **cur-dir-addr** argument is the address of the descriptor that points to the buffer area that is to receive the current directory string.

DESCRIPTION

The new directory name string is checked for correct syntax.

There is no predefined macro of the form $SETDDIR_G or $SETDDIR_S to call this service.

CONDITION VALUES RETURNED

RMS$_NORMAL	The service completed successfully.
RMS$_DIR	The directory name contains an error.
RMS$_IAL	The argument list is invalid.

SYS$SETDFPROT Set Default File Protection

The Set Default File Protection service allows you to read and write the default file protection for the process. You should restore the old default file protection specification unless you want the changed default to last beyond the exit of your image.

FORMAT

SYS$SETDFPROT *[new-def-prot-addr,] [cur-def-prot-addr]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

new-def-prot-addr

VMS usage: **file_protection**
type: **word (unsigned)**
access: **read only**
mechanism: **by reference**

A word that specifies the new default file protection specification. The **new-def-prot-addr** argument is the address of the word that specifies the desired protection. If you do not want the process-default file protection to be changed, specify the value 0.

cur-def-prot-addr

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

A word that is to receive the current default file protection specification. The **cur-def-prot-addr** argument is the address of the word that receives the current process-default protection. If you do not want the current default file protection, specify the value 0.

DESCRIPTION

There is no predefined macro of the form $SETDEFPROT_G or $SETDEFPROT_S to call this service.

CONDITION VALUES RETURNED

RMS$_NORMAL	The service completed successfully.
RMS$_IAL	The argument list is invalid.

$TRNLNM Translate Logical Name

The Translate Logical Name service returns information about a logical name.

FORMAT

SYS$TRNLNM *[attr] ,tabnam ,lognam ,[acmode] ,[itmlst]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

attr

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

Attributes controlling the search for the logical name. The **attr** argument is the address of a longword bit mask specifying these attributes. Currently, only bit 0 is used for this argument.

Each bit in the longword corresponds to an attribute and has a symbolic name. The $LNMDEF macro defines these symbolic names. To specify an attribute, specify its symbolic name or set its corresponding bit. All undefined bits in the longword must be 0.

If you do not specify this argument or specify it as 0 (no bits set), the following attribute is not used.

Attribute	Description
LNM$M_CASE_BLIND	If set, $TRNLNM does not distinguish between uppercase and lowercase letters in the logical name to be translated.

tabnam

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

Name of the table or name of a list of table names in which to search for the logical name. The **tabnam** argument is the address of a descriptor pointing to this name. This argument is required.

If the table name is not the name of a logical name table, it is assumed to be a logical name and is translated iteratively until either the name of a logical name table is found or the number of translations allowed by the system have been performed. If the table name translates to a list of logical name tables, the tables are searched in the specified order.

lognam

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

Logical name about which information is to be returned. The **lognam** argument is the address of a descriptor pointing to the logical name string. This argument is required.

acmode

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

Access mode to be used in the translation. The **acmode** argument is the address of a byte specifying the access mode. The $PSLDEF macro defines symbolic names for the four access modes.

When you specify the **acmode** argument, $TRNLNM ignores all names (both logical names and table names) at access modes less privileged than the specified access mode. The specified access mode is not checked against that of the caller.

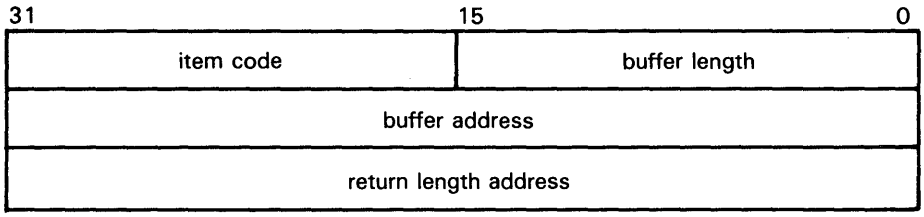
If you do not specify **acmode**, $TRNLNM performs the translation without regard to access mode; however, the translation process proceeds from the outermost to the innermost access modes. Thus, if two logical names with the same name, but at different access modes, exist in the same table, $TRNLNM translates the name with the outermost access mode.

itmlst

VMS usage: **item_list_3**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Item list describing the information that $TRNLNM is to return. The **itmlst** argument is the address of a list of item descriptors, each of which specifies or controls an item of information to be returned. The list of item descriptors is terminated by a longword of 0.

The following diagram depicts a single item descriptor.



ZK-1705-84

$TRNLNM Item Descriptor Fields

buffer length
A word specifying the number of bytes in the buffer pointed to by the **buffer address** field.

item code
A word containing a symbolic code describing the nature of the information in the buffer or to be returned to the buffer pointed to by the **buffer address** field. Each item code is described under $TRNLNM Item Codes.

buffer address
A longword containing the address of the buffer that specifies or receives the information.

return length address
A longword containing the address of a word that specifies the actual length in bytes of the information returned by $TRNLNM in the buffer pointed to by the **buffer address** field.

$TRNLNM Item Codes

LNM$_ACMODE
When you specify LNM$_ACMODE, $TRNLNM returns the access mode that was associated with the logical name at the time of its creation. The **buffer address** field in the item descriptor is the address of a byte in which $TRNLNM writes the access mode.

LNM$_ATTRIBUTES
When you specify LNM$_ATTRIBUTES, $TRNLNM returns the attributes of the logical name and the equivalence name associated with the current LNM$_INDEX value.

The **buffer address** field of the item descriptor points to a longword bit mask wherein each bit corresponds to an attribute. The $TRNLNM service sets the corresponding bit for each attribute possessed by either the logical name or the equivalence name.

The $LNMDEF macro defines the following symbolic names for these attributes.

Attribute	Description
LNM$M_CONCEALED	If $TRNLNM sets this bit, the equivalence name at the current index value for the logical name is a concealed logical name, as interpreted by RMS.
LNM$M_CONFINE	If $TRNLNM sets this bit, the logical name is not copied from a process to any of its spawned subprocesses. The DCL command SPAWN creates subprocesses.
LNM$M_CRELOG	If $TRNLNM sets this bit, the logical name was created using the $CRELOG system service.
LNM$M_EXISTS	If $TRNLNM sets this bit, an equivalence name with the specified index does exist.
LNM$M_NO_ALIAS	If $TRNLNM sets this bit, the name of the logical name cannot be given to another logical name defined in the same table at an outer access mode.
LNM$M_TABLE	If $TRNLNM sets this bit, the logical name is the name of a logical name table.
LNM$M_TERMINAL	If $TRNLNM sets this bit, the equivalence name for the logical name cannot be subjected to further (recursive) logical name translation.

LNM$_CHAIN

When you specify LNM$_CHAIN, $TRNLNM processes another item list immediately following the current item list. The LNM$_CHAIN item code must be the last one in the current item list. The **buffer address** field of the item descriptor points to the next item list.

LNM$_INDEX

When you specify LNM$_INDEX, $TRNLNM searches for an equivalence name that has the specified index value. The **buffer address** field of the item descriptor points to a longword containing a user-specified integer in the range 0 to 127.

If you do not specify this item code, the implied value of LNM$_INDEX is 0 and $TRNLNM returns information about the equivalence name at index 0.

Because a logical name may have more than one equivalence name and each equivalence name is identified by an index value, you should specify the LNM$_INDEX item code first in the item list, before specifying LNM$_STRING, LNM$_LENGTH, or LNM$_ATTRIBUTES. These item codes return information about the equivalence name identified by the current index value, LNM$_INDEX.

LNM$_LENGTH

When you specify LNM$_LENGTH, $TRNLNM returns the length of the equivalence name string corresponding to the current LNM$_INDEX value. The **buffer address** field in the item descriptor is the address of the longword in which $TRNLNM writes this length.

If an equivalence name does not exist at the current LNM$_INDEX value, $TRNLNM returns a zero to the longword pointed to by the return length field of the item descriptor.

LNM$_MAX_INDEX

Each equivalence name for the logical name has an index associated with it. When you specify LNM$_MAX_INDEX, $TRNLNM returns a value equal to the largest equivalence name index. The **buffer address** field in the item descriptor is the address of a longword in which $TRNLNM writes this value. If no equivalence names (and, therefore, no index values) exist, $TRNLNM returns a value of –1.

LNM$_STRING

When you specify LNM$_STRING, $TRNLNM returns the equivalence name string corresponding to the current LNM$_INDEX value. The **buffer address** field of the item descriptor points to a buffer containing this string. The **return length address** field of the item descriptor contains an address of a word that contains the length of this string in bytes. The maximum length of the equivalence name string is 255 characters.

If an equivalence name does not exist at the current LNM$_INDEX value, $TRNLNM returns the value 0 in the **return length address** field of the the item descriptor.

LNM$_TABLE

When you specify LNM$_TABLE, $TRNLNM returns the name of the table containing the logical name being translated. The **buffer address** field of the item descriptor points to the buffer in which $TRNLNM returns this name. The **return length address** field of the item descriptor specifies the address of a word in which $TRNLNM writes the size of the table name. The maximum length of the table name is 31 characters.

DESCRIPTION

You need read access to a shareable logical name table to translate a logical name located in that shareable logical name table.

CONDITION VALUES RETURNED

SS$_ACCVIO	The service cannot access the location or locations specified by one or more arguments.
SS$_BADPARAM	One or more arguments have an invalid value, or a logical name table name or logical name was not specified.
SS$_BUFFEROVF	The service completed successfully. The **buffer length** field in an item descriptor specified an insufficient value, so the buffer was not large enough to hold the requested data.
SS$_IVLOGNAM	The **tabnam** argument or **lognam** argument specifies a string whose length is not in the required range of 1 through 255 characters.
SS$_IVLOGTAB	The **tabnam** argument does not specify a logical name table.
SS$_NOLOGNAM	The logical name was not found in the specified logical name table or tables.
SS$_NOPRIV	The caller lacks the necessary privilege to access the specified name.

SS$_NORMAL	The service completed successfully. An equivalence name for the logical name has been found.
SS$_TOOMANYLNAM	Logical name translation of the table name exceeded the allowable depth (10 translations).

$ULKPAG Unlock Pages from Memory

The Unlock Pages from Memory service unlocks pages that were previously locked in memory by the Lock Pages in Memory ($LCKPAG) service. Locked pages are automatically unlocked and deleted at image exit.

FORMAT

SYS$ULKPAG *inadr ,[retadr] ,[acmode]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

inadr

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

Starting and ending virtual addresses of the pages to be unlocked. The **inadr** argument is the address of a 2-longword array containing, in order, the starting and ending process virtual addresses. Only the virtual page number portion of each virtual address is used; the low-order 9 bits are ignored. If the starting and ending virtual addresses are the same, a single page is unlocked.

If more than one page is being unlocked and you need to determine specifically which pages had been previously unlocked, you should unlock the pages one at a time, that is, one page per call to $ULWSET. The condition value returned by $ULWSET indicates whether the page was previously unlocked.

retadr

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

Starting and ending process virtual addresses of the pages actually unlocked by $ULKPAG. The **retadr** argument is the address of a 2-longword array containing, in order, the starting and ending process virtual addresses.

If an error occurs while multiple pages are being unlocked, **retadr** specifies those pages that were successfully unlocked before the error occurred. If no pages were successfully unlocked, both longwords in the **retadr** array contain the value –1.

acmode

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

Access mode on behalf of which the request is being made. The **acmode** argument is a longword containing the access mode. The $PSLDEF macro defines the symbols for the four access modes.

The most privileged access mode used is the access mode of the caller. To unlock any specified page, the resultant access mode must be equal to or more privileged than the access mode of the owner of that page.

DESCRIPTION

To call the $ULKPAG service, a process must have PSWAPM privilege.

CONDITION VALUES RETURNED

SS$_WASCLR	The service completed successfully. At least one of the specified pages was previously unlocked.
SS$_WASSET	The service completed successfully. All of the specified pages were previously locked.
SS$_ACCVIO	The input array cannot be read by the caller; the output array cannot be written by the caller; or a page in the specified range is inaccessible or does not exist.

$ULWSET Unlock Pages from Working Set

The Unlock Pages from Working Set service unlocks pages that were previously locked in the working set by the Lock Pages in Working Set ($LKWSET) service. Unlocked pages become candidates for replacement within the working set of the process.

FORMAT

SYS$ULWSET *inadr ,[retadr] ,[acmode]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

inadr

VMS usage: **address_range**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference–array reference or descriptor**

Starting and ending virtual addresses of the pages to be unlocked. The **inadr** argument is the address of a 2-longword array containing, in order, the starting and ending process virtual addresses. Only the virtual page number portion of each virtual address is used; the low-order 9 bits are ignored. If the starting and ending virtual address are the same, a single page is unlocked.

If more than one page is being unlocked and you need to determine specifically which pages had been previously unlocked, you should unlock the pages one at a time, that is, one page per call to $ULWSET. The condition value returned by $ULWSET indicates whether the page was previously unlocked.

retadr

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

Starting and ending process virtual addresses of the pages that were actually unlocked by $CRMPSC. The **retadr** argument is the address of a 2-longword array containing, in order, the starting and ending process virtual addresses.

If an error occurs while multiple pages are being unlocked, **retadr** specifies those pages that were successfully unlocked before the error occurred. If no pages were successfully unlocked, both longwords in the **retadr** array contain the value –1.

acmode

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

Access mode on behalf of which the request is being made. The **acmode** argument is a longword containing the access mode. The $PSLDEF macro defines the symbols for the four access modes.

The most privileged access mode used is the access mode of the caller. To unlock any specified page, the resultant access mode must be equal to or more privileged than the access mode of the owner of that page.

CONDITION VALUES RETURNED

SS$_WASCLR	The service completed successfully. At least one of the specified pages was previously unlocked.
SS$_WASSET	The service completed successfully. All of the specified pages were previously locked in the working set.
SS$_ACCVIO	The **inadr** argument cannot be read by the caller; the **retadr** argument cannot be written by the caller; or a page in the specified range is inaccessible or does not exist.
SS$_NOPRIV	A page in the specified range is in the system address space.

$UNWIND Unwind Call Stack

The Unwind Call Stack service unwinds the procedure call stack; that is, it removes a specified number of call frames from the stack. Optionally, it may return control to a new program counter (PC) unwinding the stack. The $UNWIND service is intended to be called from within a condition-handling routine.

FORMAT

SYS$UNWIND *[depadr] ,[newpc]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

depadr

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

Depth to which the procedure call stack is to be unwound. The **depadr** argument is the address of a longword value. The value 0 specifies the call frame of the procedure that was executing when the condition occurred (that is, no call frames are unwound), 1 specifies the caller of that frame, 2 specifies the caller of the caller of that frame, and so on.

If **depadr** specifies the value 0, no unwind occurs and $UNWIND returns a successful condition value in R0.

If you do not specify **depadr**, $UNWIND unwinds the stack to the call frame of the procedure that called the procedure that established the condition handler that is calling the $UNWIND service. This is the default and the normal method of unwinding the procedure call stack.

newpc

VMS usage: **address**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

New value for the program counter (PC); this value replaces the current value of the PC in the call frame of the procedure that receives control when the unwinding operation is complete. The **newpc** argument is a longword value containing the address at which execution is to resume.

Execution resumes at this address when the unwinding operation is complete.

If you do not specify **newpc**, execution resumes at the location specified by the PC in the call frame of the procedure that receives control when the unwinding operation is complete.

DESCRIPTION

The actual unwind is not performed immediately. Rather, the return addresses in the call stack are modified so that when the condition handler returns, the unwind procedure is called from each frame being unwound.

During the actual unwinding of the call stack, $UNWIND examines each frame in the call stack to see if a condition handler has been declared. If a handler has been declared, $UNWIND calls the handler with the condition value SS$_UNWIND (indicating that the call stack is being unwound) in the condition name argument of the signal array. When you call a condition handler with this condition value, that handler can perform any procedure-specific clean-up operations that may be required. After the condition handler returns, the call frame is removed from the stack.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The call stack is not accessible to the caller. This condition is detected when the call stack is scanned to modify the return address.
SS$_INSFRAME	There are insufficient call frames to unwind to the specified depth.
SS$_NOSIGNAL	No signal is currently active for an exception condition.
SS$_UNWINDING	An unwind operation is already in progress.

$UPDSEC Update Section File on Disk

The Update Section File on Disk service writes all modified pages in an active private or global section back into the section file on disk. One or more I/O requests are queued, based on the number of pages that have been modified.

FORMAT

SYS$UPDSEC *inadr ,[retadr] ,[acmode] ,[updflg] ,[efn] ,[iosb] ,[astadr] ,[astprm]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

inadr

VMS usage: **address_range**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference–array reference or descriptor**

Starting and ending virtual addresses of the pages that are to be written to the section file if they have been modified. The **inadr** argument is the address of a 2-longword array containing, in order, the starting and ending process virtual addresses. Only the virtual page number portion of each virtual address is used; the low-order 9 bits are ignored.

The $UPDSEC service scans pages starting at the address contained in the first longword specified by **inadr** and ending at the address contained in the second longword. Within this range, $UPDSEC locates read/write pages that have been modified and writes them (contiguously, if possible) to the section file on disk. Unmodified pages are also written to disk if they share the same cluster with modified pages.

If the starting and ending virtual addresses are the same, a single page is written to the section file if the page has been modified.

The address specified by the second longword may be smaller than the address specified by the first longword.

retadr

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

Addresses of the first and last pages that were actually queued for writing, in the first $QIO request, back to the section file on disk. The **retadr** argument

is the address of a 2-longword array containing, in order, the addresses of the first and last pages.

If $UPDSEC returns an error condition value in R0, each longword specified by **retadr** will contain the value –1. In this case, an event flag is not set, no AST is delivered, and the I/O status block is not written to.

acmode

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

Access mode on behalf of which the service is performed. The **acmode** argument is a longword containing the access mode. The $PSLDEF macro defines the symbols for the four access modes.

The most privileged access mode used is the access mode of the caller. A page cannot be written to disk unless the access mode used by $UPDSEC is equal to or more privileged than the access mode of the owner of the page to be written.

updflg

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

Update specifier for read/write global sections. The **updflg** argument is a longword value. The value 0 (the default) specifies that all read/write pages in the global section are to be written to the section file on disk, whether they have been modified or not. The value 1 specifies that (1) the caller is the only process actually writing the global section, and (2) only those pages that were actually modified by the caller are to be written to the section file on disk.

efn

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

Event flag to be set when the section file on disk is actually updated. The **efn** argument is a longword specifying the number of the event flag; however, $UPDSEC uses only the low-order byte.

If you do not specify **efn**, event flag 0 is used.

When you invoke $UPDSEC, the specified event flag or event flag 0 is cleared; when the update operation is complete, the event flag is set.

iosb

VMS usage: **io_status_block**
type: **quadword (unsigned)**
access: **write only**
mechanism: **by reference**

I/O status block to receive the final completion status of the updating operation. The **iosb** argument is the address of the quadword I/O status block.

When you invoke $UPDSEC, the I/O status block is cleared. After the update operation is complete, that is, when all I/O to the disk is complete, the I/O status block is written as follows:

- The first word contains the condition value returned by $QIO, indicating the final completion status.
- The first bit in the second word is set only if an error occurred during the I/O operation and the error was a hardware write error.
- The second longword contains the virtual address of the first page that was not written.

Though this argument is optional, DIGITAL strongly recommends that you specify it, for the following reasons:

- If you are using an event flag to signal the completion of the service, you can test the I/O status block for a condition value to be sure that the event flag was not set by an event other than service completion.
- If you are using the $SYNCH service to synchronize completion of the service, the I/O status block is a required argument for $SYNCH.
- The condition value returned in R0 and the condition value returned in the I/O status block provide information about different aspects of the call to the $UPDSEC service. The condition value returned in R0 gives you information about the success or failure of the service call itself; the condition value returned in the I/O status block gives you information about the success or failure of the service operation. Therefore, to accurately assess the success or failure of the call to $UPDSEC, you must check the condition values returned in both R0 and the I/O status block.

astadr

VMS usage: **ast_procedure**
type: **procedure entry mask**
access: **call without stack unwinding**
mechanism: **by reference–procedure reference or descriptor**

AST routine to be executed when the section file has been updated. The **astadr** argument is the address of the entry mask of this routine.

If you specify **astadr**, the AST routine executes at the access mode from which the section file update was requested.

astprm

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

AST parameter to be passed to the AST routine. The **astprm** argument is this longword parameter.

DESCRIPTION

The $UPDSEC service uses the calling process's direct I/O limit (DIRIO) quota in queuing the I/O request and uses the calling process's AST limit (ASTLM) quota if the **astadr** argument is specified.

Proper use of this service requires the caller to synchronize completion of the update request. You do this by first checking the condition value returned in R0 by $UPDSEC. If SS$_NOTMODIFIED is returned, the caller can continue. If SS$_NORMAL is returned, the caller should wait for the I/O to complete and then check the first word of the I/O status block for the final completion status. You can use the Synchronize ($SYNCH) service to determine whether the I/O operation has actually completed.

For a global section located in memory shared by multiple processors, only processes running on the processor that created the section can specify that global section in a call to the $UPDSEC service. Processes on another processor that attempt to update the section file will receive an error condition value indicating that the request was not performed.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully. One or more I/O requests were queued.
SS$_NOTMODIFIED	The service completed successfully. No pages in the input address range were section pages that had been modified. No I/O requests were queued.
SS$_ACCVIO	The input address array cannot be read by the caller, or the output address array cannot be written by the caller.
SS$_EXQUOTA	The process has exceeded its AST limit quota.
SS$_ILLEFC	You specified an illegal event flag number.
SS$_IVSECFLG	You specified an invalid flag.
SS$_NOTCREATOR	The section is in memory shared by multiple processors and was created by a process on another processor.
SS$_NOPRIV	A page in the specified range is in the system address space.
SS$_PAGOWNVIO	A page in the specified range is owned by an access mode more privileged than the access mode of the caller.
SS$_SHMNOTCNCT	The section is specified as being in memory shared by multiple processors, but this shared memory is not known to the system.
SS$_UNASCEFC	The process is not associated with the cluster containing the specified event flag.

$UPDSECW Update Section File on Disk and Wait

The Update Section File on Disk and Wait service writes all modified pages in an active private or global section back into the section file on disk. One or more I/O requests are queued, based on the number of pages that have been modified.

The $UPDSECW service completes synchronously; that is, it returns to the caller after writing all updated pages.

For asynchronous completion, you use the Update Section File on Disk ($UPDSEC) service; $UPDSEC returns to the caller after queuing the update request, without waiting for the pages to be updated.

In all other respects, $UPDSECW is identical to $UPDSEC. For all other information about the $UPDSECW service, refer to the documentation of $UPDSEC.

For additional information about system service completion, refer to the Synchronize ($SYNCH) service and to the *Introduction to VMS System Services*.

FORMAT

SYS$UPDSECW *inadr [,retadr] [,acmode] [,updflg] [,efn] [,iosb] [,astadr] [,astprm]*

$WAITFR Wait for Single Event Flag

The Wait for Single Event Flag service tests a specific event flag and returns immediately if the flag is set. Otherwise, the process is placed in a wait state until the event flag is set.

FORMAT

SYS$WAITFR *efn*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENT

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

Number of the event flag for which to wait. The **efn** argument is a longword containing this number; however, $WAITFR uses only the low-order byte.

DESCRIPTION

The wait state caused by this service can be interrupted by an asynchronous system trap (AST) if (1) the access mode at which the AST executes is equal to or more privileged than the access mode from which the $WAITFR service was issued and (2) the process is enabled for ASTs at that access mode.

When a wait state is interrupted by an AST and after the AST service routine completes execution, VMS repeats the $WAITFR request on behalf of the process. At this point, if the event flag has been set, the process resumes execution.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_ILLEFC	You specified an illegal event flag number.
SS$_UNASEFC	The process is not associated with the cluster containing the specified event flag.

$WAKE Wake Process from Hibernation

The Wake Process from Hibernation service activates a process that has placed itself in a state of hibernation with the Hibernate ($HIBER) service.

FORMAT

SYS$WAKE *[pidadr] ,[prcnam]*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

pidadr

VMS usage: **process_id**
type: **longword (unsigned)**
access: **modify**
mechanism: **by reference**

Process identification (PID) of the process to be awakened. The **pidadr** argument is the address of a longword containing the PID.

prcnam

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

Process name of the process to be awakened. The **prcnam** argument is the address of a character string descriptor pointing to a 1- to 15-character process name string.

The process name is implicitly qualified by the UIC group number of the calling process. For this reason, you can use the **prcnam** argument only if the process to be awakened is in the same UIC group as the calling process. To awaken a process in another UIC group, you must specify the **pidadr** argument.

If you specify neither the **pidadr** nor the **prcnam** argument, the wake request is issued for the calling process.

DESCRIPTION

Depending on the operation, the calling process may need one of the following privileges to use $WAKE:

- GROUP privilege to wake another process in the same group, unless the process has the same UIC as the calling process
- WORLD privilege to wake any other process in the system

If one or more wake requests are issued for a process not currently hibernating, a subsequent hibernate request completes immediately; that is, the process does not hibernate. No count is maintained of outstanding wakeup requests.

You can also awaken a hibernating process with the Schedule Wakeup ($SCHDWK) service.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The process name string or string descriptor cannot be read by the caller, or the process identification cannot be written by the caller.
SS$_IVLOGNAM	The specified process name string has a length of 0 or has more than 15 characters.
SS$_NONEXPR	The specified process does not exist, or you specified an invalid process identification.
SS$_NOPRIV	The process does not have the privilege to wake the specified process.

$WFLAND Wait for Logical AND of Event Flags

The Wait for Logical AND of Event Flags service allows a process to specify a set of event flags for which it wants to wait. The process is put in a wait state until all specified event flags are set, at which time $WFLAND returns to the caller and execution resumes.

FORMAT

SYS$WFLAND *efn ,mask*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

efn

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

Number of any event flag within the event flag cluster to be used. The **efn** argument is a longword containing this number; however, $WFLAND uses only the low-order byte. Specifying the number of an event flag within the cluster serves to identify the event flag cluster.

There are two local event flag clusters: cluster 0 and cluster 1. Cluster 0 contains event flag numbers 0 to 31, and cluster 1 contains event flag numbers 32 to 63.

There are two common event flag clusters: cluster 2 and cluster 3. Cluster 2 contains event flag numbers 64 to 95, and cluster 3 contains event flag numbers 96 to 127.

mask

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

Event flags for which the process is to wait. The **mask** argument is a longword bit vector wherein a bit, when set, selects the corresponding event flag for which to wait.

DESCRIPTION

The wait state caused by this service can be interrupted by an asynchronous system trap (AST) if (1) the access mode at which the AST executes is equal to or more privileged than the access mode from which the $WAITFR service was issued and (2) the process is enabled for ASTs at that access mode.

When a wait state is interrupted by an AST and after the AST service routine completes execution, VMS repeats the $WFLAND request on behalf of the process. At this point, if all the specified event flags have been set, the process resumes execution.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_ILLEFC	You specified an illegal event flag number.
SS$_UNASEFC	The process is not associated with the cluster containing the specified event flag.

$WFLOR Wait for Logical OR of Event Flags

The Wait for Logical OR of Event Flags service allows a process to specify a set of event flags for which it wants to wait. The process is put in a wait state until any one of the specified event flags is set, at which time $WFLOR returns to the caller and execution resumes.

FORMAT

SYS$WFLOR *efn ,mask*

RETURNS

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0. Condition values that this service returns are listed under CONDITION VALUES RETURNED.

ARGUMENTS

efn

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

Number of any event flag within the event flag cluster to be used. The **efn** argument is a longword containing this number; however, $WFLOR uses only the low-order byte. Specifying the number of an event flag within the cluster serves to identify the event flag cluster.

There are two local event flag clusters: cluster 0 and cluster 1. Cluster 0 contains event flag numbers 0 to 31, and cluster 1 contains event flag numbers 32 to 63.

There are two common event flag clusters: cluster 2 and cluster 3. Cluster 2 contains event flag numbers 64 to 95, and cluster 3 contains event flag numbers 96 to 127.

mask

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

Event flags for which the process is to wait. The **mask** argument is a longword bit vector wherein a bit, when set, selects the corresponding event flag for which to wait.

DESCRIPTION

The wait state caused by this service can be interrupted by an asynchronous system trap (AST) if (1) the access mode at which the AST executes is equal to or more privileged than the access mode from which the $WFLOR service was issued and (2) the process is enabled for ASTs at that access mode.

When a wait state is interrupted by an AST and after the AST service routine completes execution, VMS repeats the $WFLOR request on behalf of the process. At this point, if all the specified event flags have been set, the process resumes execution.

CONDITION VALUES RETURNED

SS$_NORMAL	The service completed successfully.
SS$_ILLEFC	You specified an illegal event flag number.
SS$_UNASEFC	The process is not associated with the cluster containing the specified event flag.

A Obsolete Services

The following table lists the obsolete system services and the current services that have replaced them. For descriptions of the obsolete services, see the *VMS Obsolete Features Manual*.

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

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

O

P

Q

R

S

T

U

V

W

Reader's Comments

VMS System Services
Reference Manual
AA–LA69A–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 ——————————————————————————————